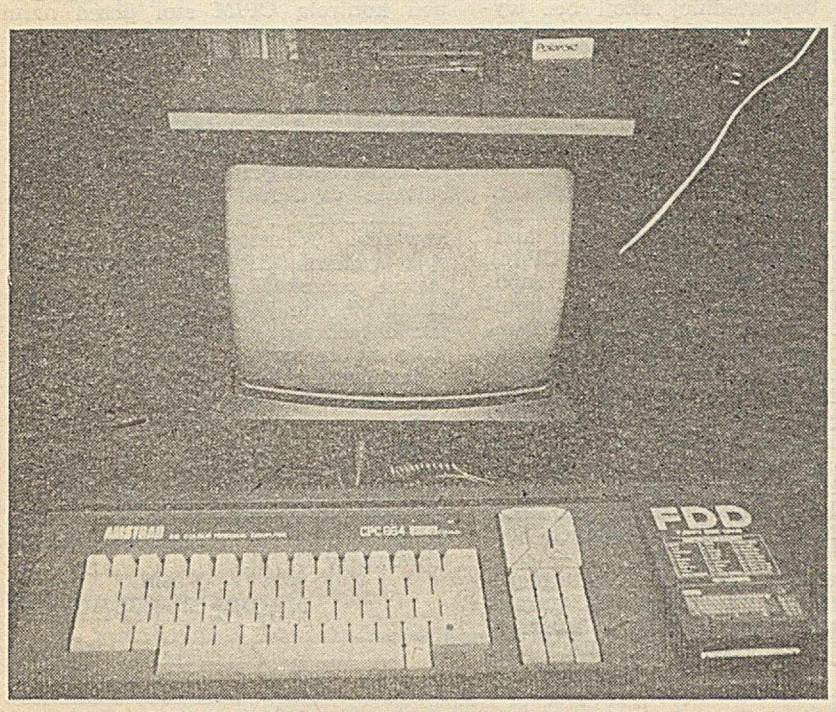



Fot. WOJCIECH JURZYK

**Publikując opis pierwszego komputerowego dziecka firmy AMSTRAD, obiecaliśmy Czytelnikom bardziej szczegółowy opis CPC 464, gdy komputer ten znajdzie się „w zasięgu" redakcji. Teraz, gdy stało się to możliwe, stanęliśmy przed dylematem — na rynku brytyjskim pojawiła się nowa, nieco rozszerzona wersja tego komputera. Ponieważ firma AMSTRAD poszła w kierunku atrakcyjnym dla sporej grupy potencjalnych odbiorców w naszym kraju, zdecydowaliśmy się na opublikowanie opisu nowej wersji. Podobnie jak poprzednio, materiał został przygotowany na podstawie informacji zaczerpniętych z pism brytyjskich, z tym jednak, że autor poniższego tekstu miał okazję wcześniej „dotknąć" CPC 664, a obecnie ów komputer ma nawet na własność.**

**Rewelacje, o których pisał autor poprzedniego materiału o AMSTRADZIE, okazały się w pełni prawdziwe. Z mankamentów, które dostrzegliśmy w dotychczasowej eksploatacji, należy wymienić:**

- **wrażliwość na niskie napięcie w sieci (w Wielkiej Brytanii jest napięcie 240 V, trudno więc mieć pretensje do komputera, że przy spadku napięcia poniżej 210 V przestaje prawidłowo funkcjonować),**
- **mała pojemność dyskietek: 170 KB zapełnia się bardzo szybko, tak więc praktycznie trzeba korzystać z dwóch napędów.**

**Z niskim napięciem w krajowej sieci można sobie poradzić stosując dostępny w sklepach stabilizator napięcia dla telewizorów. Natomiast na dyski o większej pojemności trzeba będzie prawdopodobnie jeszcze trochę poczekać.**

**Warto dodać, że w chwili, gdy przygotowujemy do druku poniższy materiał, istnieją już dwie firmy oferujące CPC 464 na rynek krajowy, za złotówki. W różnych instytucjach pracuje już kilkadziesiąt egzemplarzy tego komputera i na razie nie odnotowano żadnej awarii!**

## AMSTRAD CPC 664

Bez szumnych zapowiedzi w maju 1985 r. pojawił się na rynku angielskim nowy mikrokomputer firmy AMSTRAD — CPC 664. Podobnie jak CPC 464 w pierwszych miesiącach sprzedaży — on także długo nie stoi na półkach sklepowych. Duży popyt na CPC 664 wynika z bardzo niskiej ceny, jak na system, który zawiera: mikrokomputer wraz z wbudowanym interpreterem języka BASIC, kolorowy monitor; napęd 3-calowych dysków elastycznych wraz z systemem operacyjnym CP/M i DR LOGO. Całość kosztuje poniżej 400 funtów angielskich.

CPC 664 jest konsekwentnym rozwinięciem CPC 464. Podstawowa różnica polega na zastąpieniu w CPC 464 magnetofonu przez napęd dysków elastycznych. Ze względu na konieczność doprowadzenia napięcia 12 V, zasilającego napęd dyskowy, potrzebne jest dodatkowe gniazdo, które umieszczono w nowej wersji monitorów (monochromatyczny GT 65, kolorowy CTM 644) i w modulatorze TV oznaczonym symbolem 12V DC. W gniazdo to wtyka się drugi przewód zasilający komputer. Nowe monitory i modulator mogą współpracować z poprzednią wersją komputera, ale nie na odwrót. Umieszczenie komputera i napędu dysków w jednej obudowie spowodowało konieczność przeprojektowania obudowy. Jest ona obecnie nieco wyższa niż w CPC 464. Jednocześnie zmieniono klawiaturę na bardziej estetyczną kolorystycznie. Klawisze sterujące kursorem są większe i wygodniejsze w użyciu a klawisze numeryczne, które mogą być używane także jako funkcyjne, otrzymały dodatkowe oznaczenia — cyfra została poprzedzona literą „f"; f0, f1,...,f9.

CPC 664 ma prawie identyczną architekturę jak CPC 464. Dzięki temu oprogramowanie napisane dla 464 jest przenoszalne na 664, niestety z pewnymi wyjątkami. Niektóre programy dla 464 opracowane przez niezależnych producentów oprogramowania wykorzystują wewnętrzne procedury komputera. Ponieważ ROM został trochę poprzestawiany, to wejście w procedury „na skróty", choć przyspieszało realizację programów dla CPC 464, może teraz powodować błędy przy próbie uruchomienia takich programów na 664.

CPC 664 został wyposażony w dodatkowe gniazdo magnetofonowe, które zapewnia możliwość wczytywania zapisu programów na taśmie magnetofonowej, w sposób identyczny jak dla 464. Taśmę można więc wykorzystywać do przenoszenia programów z 464, jak również do przechowywania rzadko używanych programów lub danych. CPC 664 ma również dodatkową łączówkę do przyłączenia drugiego napędu dysków elastycznych. W 464 oba napędy przyłącza się poprzez jeden kabel wychodzący z interfejsu dyskowego, wtykanego w złącze, na które wyprowadzona jest szyna systemowa. W 664 złącze to zostało zachowane, dzięki czemu można podłączyć do komputera dodatkowe interfejsy szeregowe i równoległe lub oprogramowanie zapisane na ROM-ie (dostępne już na rynku).

CPC 664 jest wyposażony w rezydencyjny interpreter LOKOMOTIVE BASIC. Jest to nieco rozszerzona wersja w stosunku do 464. Modyfikacje dotyczą komend graficznych i obsługi błędów. Dodano ważną komendę „FILL n", której bardzo brakowało użytkownikom 464. Komenda ta pozwala na wypełnienie kolorem „n" zadeklarowanego obszaru ekranu. Poza tym podano możliwość łatwego rysowania linii przerywanych i wielobarwnych (komenda MASK). Dodatkowy parametr przy komendach rysujących linie pozwala uniknąć podwójnego rysowania końców odcinków oraz pojawiania się niezamierzonych

kropek na ekranie jako pozostałości linii. Komendy GRAPHICS PEN i GRAPHICS PAPER w nowej wersji umożliwiają tworzenie efektu „przezroczystych kolorów", dostępnego dotąd tylko dla tekstu. Dodano też komendę COPYCHAR$, która wczytuje znak z ekranu i komendę CURSOR. Są one wygodne przy pisaniu programów typu procesora tekstów. Wprowadzono również kilka innych komend ułatwiających posługiwanie się ekranem monitora. Warto też wspomnieć o nowej komendzie ON BREAK CONT — ułatwiającej pisanie programów, których nie można przerwać (w 464 działanie programu można zawsze [1]) przerwać poprzez wciśnięcie klawisza ESCAPE). Nowa komenda DERR (od „disc error") pozwala na programowe rozpoznanie przyczyny wykrytego przez maszynę błędu i podjęcie odpowiedniej akcji bez wychodzenia z programu (np. detekcja braku miejsca na dysku itp.).

Wraz z CPC 664 dostarczany jest dysk systemowy, zawierający system operacyjny CP/M (wersja 2.2) i po drugiej stronie DR LOGO (Digital Research LOGO). Pewne problemy stwarza mały obszar TPA (ang. Transient Program Area — obszar przeznaczony na programy użytkowe pod kontrolą systemu operacyjnego CP/M), który wynosi 39 K. Dlatego też zapowiedziano już następną wersję komputera o pamięci RAM 128 K. Tymczasem jednak wiele firm produkujących oprogramowanie energicznie przystąpiło do adaptacji programów pracujących pod kontrolą CP/M dla obecnych wersji AMSTRADA. I tak można już dziś nabyć kompilator języków PASCAL i TURBO PASCAL oraz FORTH, zapowiedziana jest także wersja FORTRANU. Dostępnych jest wiele programów użytkowych — procesory tekstów, programy kalkulacyjne, programy wspomagające prowadzenie przedsiębiorstwa oraz kontrolę budżetu domowego. Wszystkie ww. programy dostępne są już na dyskietkach 3-calowych. Ale można też dokupić napęd dysków 5-calowych i wtedy świat programów pracujących pod kontrolą CP/M stoi przed nami otworem, niezależnie od firm przenoszących oprogramowanie na dyski 3-calowe.

Dużą zaletą CPC 664 jest dobry podręcznik. Przy opracowywaniu podręcznika dla 664 skorzystano z uwag na temat podręcznika dla 464. W rezultacie użytkownik otrzymuje podręcznik zwięzły, jasny i łatwy w użytkowaniu. Niestety na szczegółowy opis systemu CP/M i LOGO trzeba jeszcze poczekać. Firma zapowiada, że ukażą się one jeszcze latem 1986 r., a jak dotychczas punktualnie wywiązywała się ze swoich zapowiedzi.

Reasumując — AMSTRAD CPC 664 jest nie tylko połączeniem CPC 464 z napędem dysków elastycznych, ale dzięki wprowadzonym rozszerzeniom jest to praktycznie nowy komputer domowy, który dzięki atrakcyjnej cenie znajduje wielu nabywców. Jest to najtańszy obecnie system wyposażony firmowo w CP/M.

**MACIEJ DORUCHOWSKI**

Warszawa

[1]) W CPC 464 istnieje specjalna komenda „chronionego" zapisu programów, zabezpieczająca przed kopiowaniem przez osoby niepowołane. Po wprowadzeniu do komputera, program taki samoczynnie startuje i nie daje się przerwać. Nowa komenda została prawdopodobnie przewidziana jako zabezpieczenie przed mało wprawnym operatorem (np. programy edukacyjne pisze się również dla dzieci w wieku 3 lat, a mają one zwyczaj „próbowania" wszystkich możliwych klawiszy) — przyp. AJP.

---

**Po dłuższej (niestety!) przerwie wznawiamy publikację materiałów o grafice komputerowej. Tym co już zdążyli zapomnieć przypominamy, że Antoni Urban dzieli się doświadczeniami zdobytymi przy realizacji implementacji standardu GKS (Graphical Kernel System) na IBM PC. Ponieważ implementacja została zrealizowana w języku C, publikowane uwagi można wykorzystać i dla innych mikrokomputerów (oczywiście wyposażonych w możliwości graficzne), jeżeli tylko dysponujemy kompilatorem języka C.**

## Wielolinia

Tytuł artykułu jest dosłownym tłumaczeniem angielskiego terminu **polyline**. Wielolinia jest podstawowym prymitywem reprezentacji graficznej przyjętej przez twórców normy GKS. Jest to tak charakterystyczne podejście do problemu rysowania linii, że uzasadnione stało się tworzenie nowych terminów językowych. W innych, stosowanych wcześniej językach graficznych, rozkaz rysowania linii wymagał podania współrzędnych początku i końca odcinka lub samego końca, gdy za początek przyjmowane było aktualne położenie domyślnego znacznika. W przypadku GKS wystarczy podać liczbę punktów i nazwę tablicy zawierającej współrzędne kolejnych punktów wyznaczających kontur. Kontur ten składa się z odcinków linii prostej, jest więc linią łamaną. Przed przystąpieniem do rysowania należy wyodrębnić wszystkie kontury, dla każdego zadeklarować tablicę i wypełnić współrzędnymi punktów wyznaczających kontur. Na przykład, narysowanie ramki ograniczającej obszar rysowania wymaga deklaracji tablicy o wymiarach 5 × 2 i wpisania do niej kolejno: $(x_{min}, y_{min})$, $(x_{min}, y_{max})$, $(x_{max}, y_{max})$, $(x_{max}, y_{min})$, $(x_{min}, y_{min})$.

Na tablicach można wykonywać takie działania, jakie wprowadza się dla konturów, a więc: translacja, obrót, jednokładność, powinowactwo, wyznaczanie ekwidystanty, sklejanie, obcinanie i inne. Wymienione działania należy oprogramować we własnym zakresie, ponieważ norma GKS dostarcza narzędzi do animacji rysunku tylko w przypadku segmentów, o których będzie jeszcze mowa.

Wielolinia może być rysowana na dowolnych, przyłączonych do komputera urządzeniach graficznych (monitor, ploter, drukarka) z uwzględnieniem atrybutów, do których należą: kolor linii, rodzaj linii (ciągła, kreskowana, kropkowana, przerywana) oraz długość części składowej linii przerywanej. W zależności od urządzenia graficznego kolor linii oznacza: kod koloru linii oraz tła (monitor lub drukarka — kolorowe), stopień szarości (monitor lub drukarka — monochromatyczne) albo numer pióra (ploter).

Cykl artykułów dotyczących normy GKS znalazł się na łamach mikroKLANU, ponieważ jest opisem nie tyle samej normy GKS, lecz jej implementacji dla mikrokomputerów oraz problemów związanych z adaptacją jądra graficznego dla małego sprzętu informatycznego. Oprogramowanie GKS napisane zostało w języku C, ale może być traktowane jak biblioteka procedur w języku wewnętrznym i wykorzystywane w dowolnym języku programowania. Oczywiście najbardziej naturalne jest stosowanie tego samego języka, to znaczy C. Ograniczeniem narzuconym przez język programowania jest długość nazw procedur. Autor proponuje stosowanie jednej tablicy o wymiarach n × 2, zamiast dwóch wektorów o długości n. Przy wymienionych odstępach od normy, rozkaz wywołania prymitywu wielolinia ma następującą składnię:

**Polyline (n, p);**

gdzie int n — liczba punktów, float p[ ][2] — współrzędne kolejnych punktów.

Należy podkreślić, że współrzędne podawane są w układzie rysunku wprawdzie wygodnym dla użytkownika, lecz odbiegającym od układu urządzenia graficznego.
Rozkazy deklaracji atrybutów mają postać:

**Set_ltype (lt);**

gdzie: char lt — rodzaj linii (lt = 0 — ciągła; lt = 1 — kreska; lt = 2 — kropka-kropka; lt = 3 — kreska-kropka)

**Set_lcolour (lc);**

gdzie: char lc — kod koloru linii oraz tła (lc = 0 — czarny, lc = 1 — niebieski, lc = 2 — zielony, lc = 3 — niebies-

ko-zielony, lc = 4 — czerwony, lc = 5 — fioletowy, lc = 6 — żółty, lc = 7 — biały)

**Set_lsize (ls);**

gdzie: float ls — długość części składowej linii przerywanej.

Atrybuty prymitywu wielolinia mogą być deklarowane sukcesywnie lub — po wcześniejszym przypisaniu zestawów atrybutów numerowi indeksu — przełączane. Zazwyczaj rysunki wykonywane są przy zastosowaniu skończonej liczby reprezentacji atrybutów. Do przygotowania i przełączania indeksu służą rozkazy:

**Line_representation (li, lt, ls, lc);**

gdzie: int li — numer indeksu; char lt — rodzaj linii; float ls — długość części składowej linii przerywanej; char lc — kod koloru
oraz

**Set_lindex (li);**

gdzie: int li — numer indeksu

Realizacja implementacji prymitywu wielolinia dla mikrokomputera wymagała rozwiązania następujących dwu problemów:

1. Jak uwzględnić rodzaj linii i długość części składowej linii przerywanej?

2. Jak rysować linię na urządzeniach graficznych bez interpolatora liniowego?

**Problem 1** rozwiązano przez zakodowanie rodzaju linii w bajcie, w którym bit 1 oznacza **rysuj**, a bit 0 — **nie rysuj**. I tak poszczególnym rodzajom linii odpowiadają następujące kody:

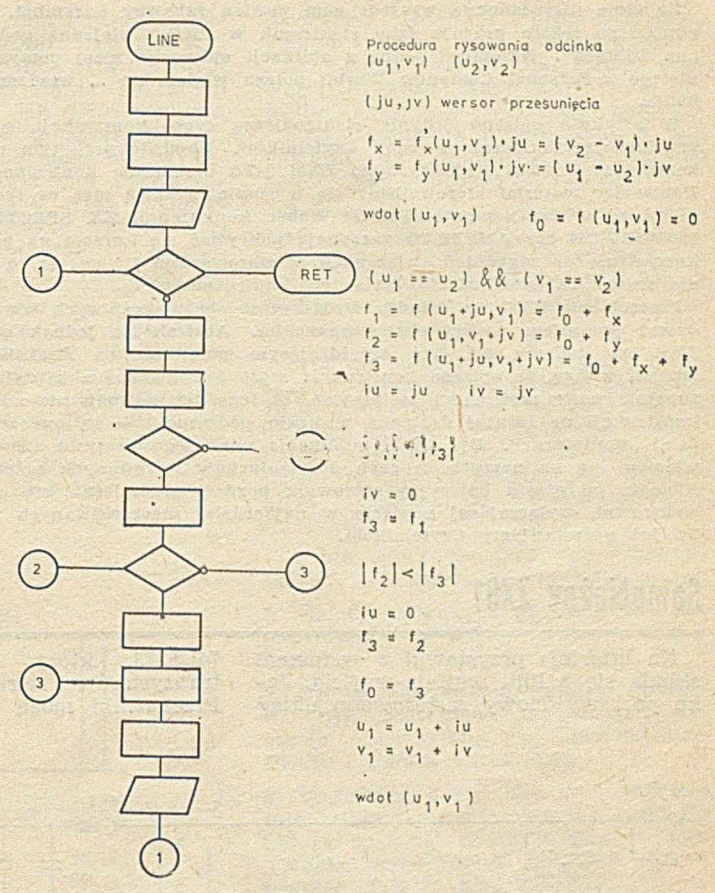
| | | |
|---|---|---|
| linia ciągła | -------- | $11111111_B$ |
| kreska | ---- | $11110000_B$ |
| kropka-kropka | - - | $10001000_B$ |
| kreska-kropka | --- - | $11100100_B$ |

Rozmiar linii uwzględniany jest przez wielokrotne (zależne od deklaracji długości części składowej linii przerywanej) testowanie tego samego bitu kodu rodzaju linii. Takie rozwiązanie problemu daje możliwość uwzględniania rodzaju linii wewnątrz procedury generacji linii.

**Problem 2** rozwiązano wykorzystując algorytm przedstawiony na załączonym rysunku schematu blokowego. Algorytm przytoczony jest dla osób, które swój sprzęt mikrokomputerowy chcą wyposażyć w interpolator linii. Podana metoda jest szczególnym przypadkiem algorytmu generacji krzywych drugiego stopnia, zadanych równaniem uwikłanym:

$$f(x, y) = 0$$

Stosowany jest tu wzór Taylora dla funkcji dwu zmiennych rzeczywistych:




Schemat blokowy algorytmu

$$[1]\quad f(x+\Delta x, y+\Delta y) = f(x,y) + f'_x(x,y)\Delta x + f'_y(x,y)\Delta y + \frac{1}{2}(f''_{xx}(x,y)\Delta x^2 + 2f''_{xy}(x,y)\Delta x\Delta y + f''_{yy}(x,y)\Delta y^2)$$

$$[2]\quad f'_x(x+\Delta x, y+\Delta y) = f'_x(x,y) + f''_{xx}(x,y)\Delta x + f''_{xy}(x,y)\Delta y$$

$$[3]\quad f'_y(x+\Delta x, y+\Delta y) = f'_y(x,y) + f''_{xy}(x,y)\Delta x + f''_{yy}(x,y)\Delta y$$

Za pomocą wzoru [1] obliczane są wartości funkcji f dla trzech możliwych kierunków przesunięć ($f_1, f_2, f_3$). Spośród tych liczb wybierana jest ta, której wartość bezwzględna jest najmniejsza, co zapewnia optymalne przemieszczenie. Po dokonaniu wyboru modyfikowane są wartości pochodnych cząstkowych według wzorów [2] i [3].
Algorytm ten wykorzystany zostanie jeszcze przy omawianiu interpolatora kołowego (prymityw G.D.P.).

**ANTONI URBAN**
Computer Studio Kajkowscy
Gdynia

---

Firma IBM wypuściła nowy model komputera osobistego o nazwie IBM JX. Wszystko wskazuje, że jest to następca nieudanego modelu IBM PCjr. Dosyć zaskakujący może wydawać się fakt, że nowy komputer trafił na razie tylko na rynek... australijski. Być może jest to generalna próba przed wprowadzeniem go do Europy Zachodniej i USA. Próba o tyle sensowna, że komputer montowany jest w Japonii (gdzie oczywiście sprzedawana jest japońska wersja — mało użyteczna dla Europejczyków). Model podstawowy zawiera tylko 64 KB pamięci RAM, ale za to aż 256 KB ROM. Wynika to z faktu, że najprostsza wersja nie jest wyposażona w dyski elastyczne. Razem z monitorem kolorowym kosztuje ona 2115 dol. australijskich. Poszerzona wersja ma 128 KB RAM i napęd dysków elastycznych 3,5". Na dyskietce można zapisać 180 KB. Najbardziej rozbudowana wersja ma 256 KB RAM i dwa napędy 3,5". Kosztuje ona 3365 dol. australijskich. Konstrukcja komputera umożliwia rozszerzenie pamięci do 512 KB RAM. Możliwe jest też dołączenie dodatkowego napędu 5,25" (596 dol.), co pozwala uzyskać przenośność oprogramowania i danych z IBM PC. Z punktu widzenia firmy IBM, nowy komputer jest tani, tyle że w domowych zastosowaniach nalepka IBM nie jest taka ważna.

**Zupełnie nieświadomie wyszedł nam pewien zabawny karambol. Otóż autor poniższego tekstu pracuje jako elektronik w PKP i niejedną godzinę spędził „na kółkach". Natomiast jedna z aplikacji opisanego niżej pomysłu (zaczerpniętego z Personal Computer World) polega właśnie na ... wsadzeniu ZX81 na kółka.**

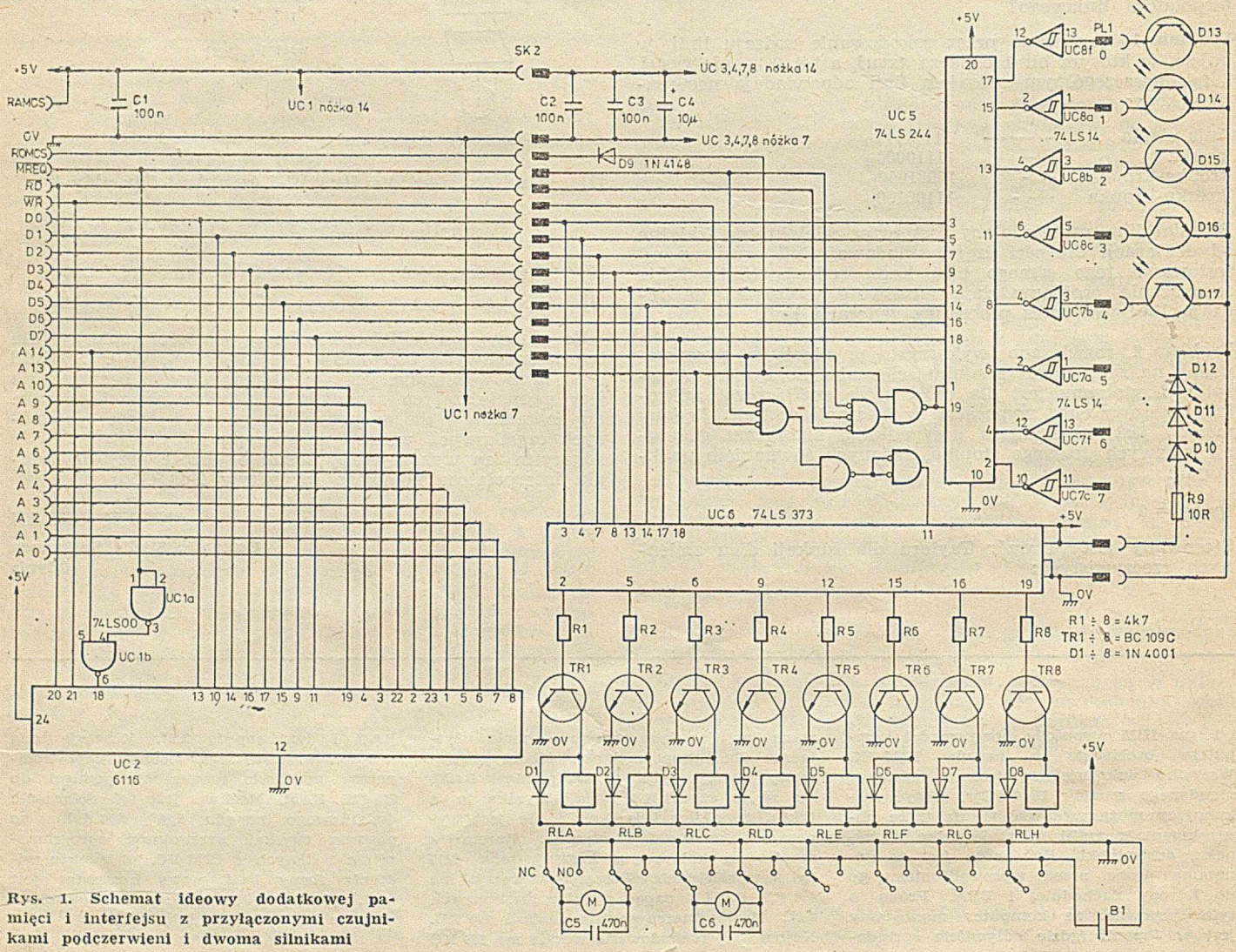
**W artykule opisano jedynie elektroniczną część konstrukcji, pozostawiając problemy mechaniczne inwencji Czytelników. Chodziło przy tym raczej o pokazanie możliwości wykorzystania ZX81 jako urządzenia kontrolno-sterującego. Dobierając materiał kierowaliśmy się w sumie tą samą ideą co redakcja PCW — nawet u nas komputerki ZX81 wobec konkurencji ZX SPECTRUM, COMMODORE C64 czy AMSTRAD zaczynają pokrywać się kurzem na półkach. Wykorzystanie do zagadnień związanych ze sterowaniem i kontrolą to może ostatnia szansa na przedłużenie życia tego mikrokomputera.**

**Przerobienie ZX81 na dziadka stróża może się w krajowych warunkach wydawać pomysłem nazbyt ekstrawaganckim. Abstrahując jednak od tego, czy ZX81 na kółkach ma być objeżdżającym pomieszczenia strażnikiem, warto zwrócić uwagę, że opisana konstrukcja może być zasilana z baterii, co stwarza zupełnie nowe możliwości zastosowań. Z tego też powodu nie adaptowaliśmy konstrukcji na łatwiej dostępne w kraju podzespoły — są one znacznie bardziej „prądożerne". Być może publikacja poniższego pomysłu zaowocuje pojawieniem się na naszych drogach np. maluchów sterowanych przez komputer. A może po prostu bateryjny sterownik pozwoli uniezależnić się od humorów elektrowni, wyłączającej napięcie w najbardziej nieoczekiwanych momentach? To już pozostawiamy Czytelnikom.**

## Samobieżny ZX81

Na interfejs przystawki z systemem składa się 8 linii wejścia-wyjścia. Jako port wyjściowy zastosowano układ 74LS373, który za pośrednictwem tranzystorów steruje przekaźnikami. Przekaźniki mogą być wykorzystywane do włączania małych silników, brzęczyków itp. Zapis danych do portu można zrealizować wykonując instrukcję POKE 9000, dana. Wpisana pod adres 9000 (dziesiętnie) dana powinna być traktowana w sposób binarny, tzn. wpisanie 1 włącza przekaźnik sterowany bitem D1 (pozostałe pozostają wyłączone), wpisanie 5 włącza przekaźniki sterowane bitami D1 i D4 itp. Jako port wejściowy zastosowano układ 74LS244 umożliwiający odczyt stanu logicznego na 8 liniach.

Interfejs przystawki można umieścić w obszarze adresowym pamięci między 8 a 16 K lub 40 a 48 K. Najwygodniej jednak jest wykorzystać adres 9000 (dziesiętnie). Pod tym adresem znajduje się „cień" firmowej pamięci ROM minikomputera. Obszar z „cieniem" pamięci ROM nie jest jednak pusty. Operacje odczytu interfejsu byłyby więc zafałszowane. Aby tego uniknąć linia ROMCS jest przyłączona do linii adresowej A13 przez diodę D9. Dzięki temu pojawienie się jakiegokolwiek adresu, w którym linia A13 szyny adresowej sterowana jest poziomem wysokim, podnosi również poziom sygnału na ROMCS (do-




**Rys. 1. Schemat ideowy dodatkowej pamięci i interfejsu z przyłączonymi czujnikami podczerwieni i dwoma silnikami**

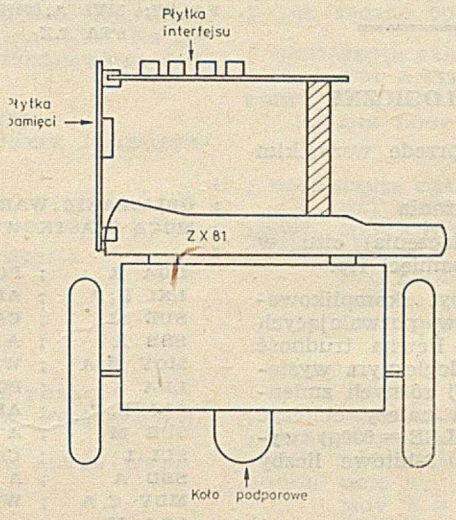
tyczy to każdego adresu z obszaru 8—16 K, zakrywanego przez ROM systemowy).

Komputer wpisuje dane do pamięci ROM sygnałami MREQ/ i WR/, aktywnymi w stanie logicznym 0. Operacja dotyczy obszaru ROM systemowego (lub jego „cienia") gdy linia A14 sterowana jest poziomem niskim. Aby przygotować układ 74LS373 (UC6) do zapisu, należy jeszcze stan logiczny 1 z wyjścia UC3b przemnożyć logicznie ze stanem 1 linii adresowej A13. Daje to gwarancję, że operacja nie dotyczy obszaru zajętego przez ROM systemowy. Z wyjścia UC4a podawany jest poziom wysoki na wejście E zatrzasku UC6. Przejście linii MREQ z powrotem na poziom wysoki wywołuje opadające zbocze na wyjściu bramki UC4a i zatrzaskuje nowe dane na liniach wyjściowych UC6. Linie te poprzez tranzystory TR1 ... 7 sterują przekaźnikami. Diody przyłączone równolegle do uzwojeń przekaźników zabezpieczają tranzystory przed przepięciami wstecznymi, występującymi przy wyłączaniu prądu w cewce.

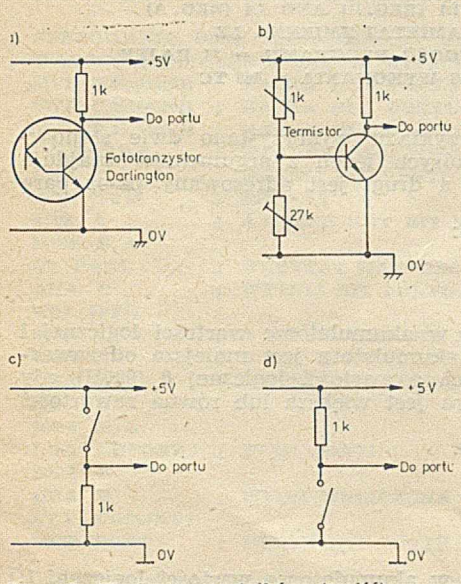
Układ wyboru portu wejściowego jest zasadniczo taki sam jak dla portu wyjściowego. Różnica polega na tym, że układ 74LS244 (UC5) pobudzany jest wtedy, gdy komputer chce czytać dane. Układ wyboru umożliwia bezpośrednie przyłączenie linii wyjściowych UC5 do szyny danych ZX81. Inwertery Schmitta (UC7 i UC8) służą do „poprawiania" zboczy sygnałów wejściowych.

Standardowa w ZX81 pamięć 1 K nie wystarcza nawet do umieszczenia w miarę prostych programów kontrolno-sterujących. Zastosowanie modułu 16 K RAM nie jest zbyt dobre ze względu na znaczny wzrost zużycia energii. Wygodnie jest zastosować tu pamięć CMOS (układ 6116 2 K × 8 CMOS RAM (UC2). W opisywanym rozwiązaniu obniża ona nawet zużycie energii przez sam ZX81. UC2 jest wybierany wtedy, gdy linie A14 i MREQ/ są równocześnie w stanie logicznym 0, tzn. identycznie jak ULA wybiera wewnętrzną pamięć w ZX81. Dlatego też linia RAMCS nie może być wykorzystywana — została ona przyłączona do napięcia +5 V (stan logiczny 1), co powoduje odcięcie wewnętrznej pamięci RAM komputera.

Do portu wejściowego można dołączyć różnego rodzaju czujniki tak, aby dostarczane przez nie informacje mogły być wprowadzane do komputera (rys. 2). Układ może, na przykład, reagować na światło (rys. 2a) lub temperaturę (rys. 2b), albo też wprowadzać sygnały z przełącznika zależnie od przyjętej dodatniej lub ujemnej logiki (rys. 2c i 2d).

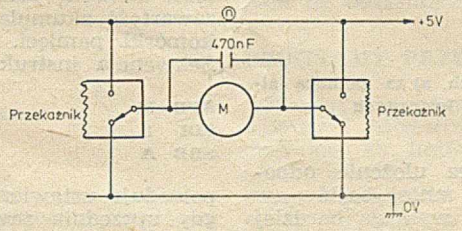



**Rys. 3. Samobieżny wózek sterowany ZX81 (zasilanie z baterii 9 V), każde koło jest sterowane oddzielnym silnikiem**




**Rys. 2. Warianty czujników (wejść): a) czujnik świetlny, b) czujnik temperatury, c) i d) czujniki przełącznikowe**

ZX81 z opisanym układem można zamontować na wózku (rys. 3). Przekaźniki wyjściowe mogą wtedy sterować silnikami napędzającymi kółka (rys. 4) i w ten sposób ZX81 może poruszać się zależnie od zewnętrznych sygnałów (temperatura, światło itp.) — stając się czymś w rodzaju robota lub sławnego niegdyś cybernetycznego żółwia. Przekaźniki wyjściowe sterują silnikami napędzającymi dwa koła (rys. 3), a sygnały wejściowe dostarczane są przez czujniki podczerwieni. Jeżeli wszystkie okna i drzwi zabezpieczymy źródłami promieniowania podczerwonego, to „zmotoryzowany" ZX81 może przyjąć na siebie obowiązki... stróża. Nasz wartownik objeżdża cyklicznie wszystkie pomieszczenia (zgodnie z programem), a gdy natrafia na „nieprawidłowe" źródło podczerwieni, uruchamia alarm.




**Rys. 4. Sterowanie silnika**

Jest to oczywiście jedno z zastosowań ruchomego ZX81. Zmieniając odpowiednio czujniki wejściowe, a także urządzenia, którymi steruje ZX81, można wymyśleć dla niego wiele ciekawych zajęć.

Program kontrolno-sterujący napisany w języku maszynowym (ASSEMBLER Z80) powinien zmieścić się w ok. 500 bajtach. Powiększona pamięć RAM umożliwa napisanie programu nawet w języku BASIC, pod warunkiem oszczędnego gospodarowania miejscem.

A oto kilka prostych rad zmierzających do skracania programu:

**1. Nazwy zmiennych powinny być jak najkrótsze — każda litera to jeden bajt.**

**2. Często powtarzane numery powinny być zastąpione zmiennymi, np. często powtarzany w instrukcjach PEEK i POKE adres 9000.**

**3. Wszędzie, gdzie to jest możliwe (nie zmienia się odczytywana wartość) należy zastąpić IF PEEK 9000 = xxx THEN... przez LET D = PEEK 9000 i używać IF D = xxx THEN...**

Przy pewnej wprawie w programowaniu takie lub podobne metody oszczędzania pamięci można wymyśleć samemu.

**ROLAND HOJCZAK**
Ośrodek Badań Stanu Torów PKP
Warszawa

---

Nowe mikrokomputery firmy AMSTRAD 6128 i 8256 mają więcej pamięci RAM, niż wynosi obszar adresowy mikroprocesora Z80. Jest to możliwe dzięki zastosowaniu pewnej sztuczki: pamięć podzielona jest na strony, a szyny mikroprocesora przełączane są pomiędzy tymi stronami. Jak można się domyślać z nazwy, model 6128 zawiera 128 KB RAM, a model 8256 ma 256 KB. Użytkowników tego ostatniego zapewne ucieszy informacja, że jego pamięć można jeszcze dwukrotnie powiększyć. Otóż po zdjęciu obudowy okazuje się, że komputer zawiera... 8 pustych podstawek. Wtajemniczeni twierdzą, iż początkowo firma AMSTRAD projektowała model 8128. W tym czasie układy pamięci DRAM o pojemności 256 Kbitów tak staniały (ok. 1,5 dol. za sztukę), że bardziej opłacalne było zamontowanie 8 układów 256 Kb niż 16 układów 64 Kb! W ten sposób zostało wolne miejsce, które po wykonaniu kilku połączeń może zostać wykorzystane dla rozszerzenia pamięci. Dodatkowa pamięć może zostać wykorzystana na tzw. RAM-disc.

**Skromne łamy mikroKLANU powstrzymują nas od wykładania podstaw języka ASSEMBLER. Nie oznacza to jednak, że pozostawiamy początkujących zupełnie bez opieki. Liczymy, że Czytelnicy potrafią znaleźć w jakiejś książce znaczenie poszczególnych rozkazów ASSEMBLERA 8080. Poniższy dwuczęściowy materiał pokazuje natomiast jak programować. Prezentowane procedury nie były optymalizowane ani pod kątem czasu realizacji, ani miejsca zajmowanego w pamięci — pozostawiamy to już jako ćwiczenie wprawiające. Chodziło nam o klarowne ukazanie zapisu algorytmów realizujących pewne użyteczne funkcje. O ile procedury realizujące funkcje logiczne przedstawiają przykładowe rozwiązania pewnych szczególnych problemów, o tyle algorytmy opóźnień czasowych i konwersji kodów mogą zostać włączone do zestawu procedur bibliotecznych, podobnie jak opisywane w poprzednim odcinku procedury arytmetyczne.**

# Przykłady procedur w języku ASSEMBLER 8080 (2)

## PROGRAMY REALIZUJĄCE FUNKCJE LOGICZNE

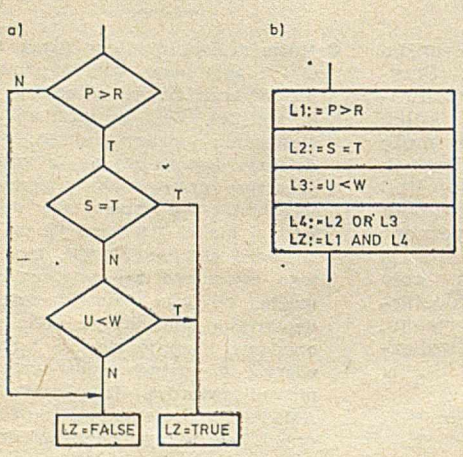
Funkcje logiczne wykorzystywane są przede wszystkim w dwóch przypadkach:

- dla wyliczenia wartości logicznej wyrażenia
- dla wysterowania (włączenia lub wyłączenia) bitu w porcie zewnętrznym, rejestrze, komórce pamięci itp.

Pierwszy przypadek nie wydaje się zbyt skomplikowany. Mikroprocesor 8080 ma szereg rozkazów, pozwalających na realizację każdej operacji logicznej. Pewna trudność pojawia się wówczas, gdy w wyrażeniu logicznym występuje wiele operatorów logicznych i relacji różnych zmiennych. Rozpatrzmy przypadek, w którym należy obliczyć wartość logiczną (TRUE = $0FF_H$ lub FALSE = $000_H$) wyrażenia logicznego, w którym występują 8-bitowe liczby całkowite bez znaku.

LZ = P > R AND (S = T OR U < W)

Wartość zmiennej logicznej Z można obliczyć korzystając z jednego ze sposobów przedstawionych na rysunku 1.




**Rys. 1. Sposoby obliczania funkcji logicznych a) za pomocą algorytmu; b) za pomocą cząstkowych operacji logicznych**

Obliczanie wartości wyrażenia poprzez ułożenie odpowiedniego algorytmu zajmuje z reguły mniej bajtów w pamięci i wykonuje się szybciej, jednak wymaga bardziej wnikliwej analizy wyrażenia i dlatego częściej stosowane jest przez programistów, piszących program w języku symbolicznym, niż przez kompilatory języków wyższego poziomu, zamieniające postać wyrażenia na kod maszynowy. Przykłady programów realizujących obie metody obliczania wyrażeń logicznych przedstawiono poniżej.

```
; OBLICZANIE WARTOŚCI WYRAŻENIA LOGICZNEGO ZA POMOCĄ ALGORYTMU
;
        LDA  R          ; POBIERZ ZMIENNA R
        LXI  H,P        ; ADRES ZMIENNEJ P
        CMP  M          ; POROWNAJ P I R
        JNC  FALSE
        LDA  S          ; POBIERZ ZMIENNA S
        LXI  H,T        ; ADRES ZMIENNEJ T
        CMP  M          ; POROWNAJ S I T
        JZ   TRUE
        LDA  U          ; POBIERZ ZMIENNA U
        LXI  H,W        ; ADRES ZMIENNEJ W
        CMP  M
        JNC  FALSE
TRUE:   MVI  A,0FFH     ; WARTOŚĆ TRUE
        STA  LZ         ; ZMIENNA LZ ROWNA TRUE
        ...
FALSE:  MVI  A,000H     ; WARTOŚĆ LOGICZNA FALSE
        STA  LZ         ; ZMIENNA LZ ROWNA FALSE
                        ; DŁUGOŚĆ PROGRAMU = 40 BAJTOW
                        ; CZAS WYKONANIA = 57 TC do 140 TC
;
; OBLICZANIE WARTOŚCI WYRAŻENIA LOGICZNEGO ZA POMOCĄ CZĄSTKOWYCH OPERACJI LOGICZNYCH
;
    LDA  R      ; POBIERZ ZMIENNA R
    LXI  H,P    ; ADRES ZMIENNEJ P
    SUB  M      ; CARRY = 1 GDY P WIEKSZE OD R
    SBB  A      ; A = 0FFH GDY P WIEKSZE OD R
    MOV  B,A    ; WYNIK CZASTKOWY L1
    LDA  S      ; POBIERZ ZMIENNA S
    LXI  H,T    ; ADRES ZMIENNEJ T
    SUB  M      ; A = 000H GDY S ROWNE T
    SUI  1      ; CARRY = 1 GDY S ROWNE T
    SBB  A      ; A = 0FFB GDY S ROWNE T
    MDV  C,A    ; WYNIK CZASTKOWY L2
    LDA  U      ;
    LXI  H,W    ; ADRES ZMIENNEJ W
    SUB  M      ; CARRY = 1 GDY U MNIEJSZE OD W
    SBB  A      ; A = 0FFH GDY U MNIEJSZE OD W
    ORA  C      ; L4 (REG. A) = L2 (REG. C) OR L3 (REG. A)
    ANA  B      ; A = L1 (REG. B) AND L4 (REG. A)
    STA  LZ     ; ZAPAMIETAJ ZMIENNA LZ
                ; DŁUGOŚĆ PROGRAMU = 33 BAJTY
                ; CZAS WYKONANIA = 140 TC
```

W drugiej części przykładu wykorzystano dwie standardowe operacje, w których jeden z argumentów znajduje się w akumulatorze, a drugi jest adresowany przez parę rejestrów HL.
Sekwencja instrukcji:

```
SUB M
SBB A
```

powoduje ustawienia w akumulatorze wartości logicznej 1 (0FFH) gdy wartość akumulatora jest mniejsza od zawartości komórki pamięci, a wartości logicznej 0 (000H) gdy zawartość akumulatora jest większa lub równa zawartości komórki pamięci.
Sekwencja instrukcji:

```
SUB M
SUI 1
SBB A
```

powoduje ustawienie w akumulatorze wartości logicznej 1, gdy uprzednia zawartość akumulatora jest równa zawartości komórki pamięci M (o adresie zawartym w rejes-

trach H i L), a w przeciwnym przypadku ustawienie w akumulatorze wartości logicznej 0.

Przypadek wykorzystania funkcji logicznych dla ustalenia lub kasowania wybranego bitu stwarza więcej kłopotów z uwagi na brak w procesorze 8080 rozkazów operujących na bitach. Z braku innych możliwości muszą być tu wykorzystane w odpowiedni sposób rozkazy operujące na całych bajtach. Dla „zapalenia" („zgaszenia") wybranego bitu w odpowiednim rejestrze, porcie lub komórce pamięci może być wykorzystany rozkaz sumy logicznej (iloczynu logicznego) z maską, w której jest „zapalony" („zgaszony") tylko wybrany bit. Ilustruje to poniższy przykład:

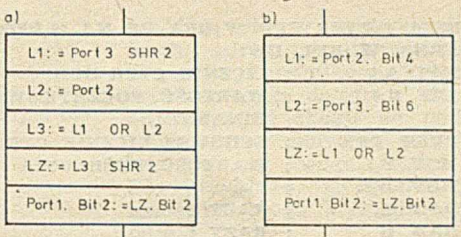
```
; ROZKAZY ZAPALAJĄCE I GASZĄCE CZWARTY (A3) BIT
  AKUMULATORA
;
  ORI 00001000B   ; ZAPAL BIT 4 AKUMULATORA
  .....
  ANI 11110111B   ; ZGAŚ BIT 4 AKUMULATORA
```

Często w programach realizujących sterowanie sekwencyjne stan wybranej linii (bitu) portu wejściowego jest funkcją stanów linii (bitów) portów wejściowych. W tym wypadku przed wysterowaniem odpowiedniej linii należy wyliczyć wartość logiczną, jaką należy jej przypisać. Podobnie jak w przypadku obliczania wartości logicznych wyrażeń, można to zrealizować na kilka sposobów. Na rysunku 2 oraz w poniższym przykładzie przedstawiono dwa sposoby rozwiązania problemu:

```
  PORT1—BIT2 = PORT2—BIT4 OR PORT3—BIT6
; STEROWANIE BITU WYJŚCIOWEGO POPRZEZ PRZESUWA-
; NIE BITÓW WEJŚCIOWYCH
  MVI B,00001000B  ; MASKA DLA PORTU 2
  MVI C,00100000B  ; MASKA DLA PORTU 3
  MVI D,11111101B  ; MASKA DLA PORTU 1
  IN PORT 2        ; WCZYTAJ WARTOŚĆ PORTU 2
  ANA B            ; WYTNIJ BIT PORTU 2
  MOV B, A
  IN PORT 3        ; WCZYTAJ PORT 3
  ANA C            ; WYTNIJ BIT PORTU 3
  RRC
  RRC              ; PRZESUŃ NA POZYCJE PORTU 2
  ORA B            ; SUMA BITÓW PORTÓW WEJŚCIOWYCH
  RRC
  RRC              ; PRZESUN NA POZYCJE PORTU WYJŚ-
                     CIOWEGO
  MOV B, A
  LDA ZMPORT 1     ; STAN PORTU 1
  ANA D            ; WYKASUJ BIT PORTU 1
  ORA B            ; WPISZ WYLICZONA WARTOŚĆ
  STA ZMPORT 1
  OUT PORT 1       ; USTAW BIT PORTU 1

; STEROWANIE BITU WYJŚCIOWEGO POPRZEZ WYLICZENIE
; WARTOŚCI LOGICZNEJ BITÓW WEJŚCIOWYCH
  MVI B,00001000B  ; MASKA DLA PORTU 2
  MVI C,00100000B  ; MASKA DLA PORTU 3
  MVI D,11111101B  ; MASKA DLA PORTU 1
  IN PORT 2        ; WCZYTAJ WARTOŚĆ PORTU 2
  ANA B            ; WYTNIJ BIT PORTU 2
  ADI 0FFH         ; BIT = 0 ?
  SBB A            ; A = 0FFH GDY BIT ROWNY 1
  MOV B, A
  IN PORT 3        ; WCZYTAJ WARTOŚĆ PORTU 3
  ANA C            ; WYTNIJ BIT PORTU 3
  ADI 0FFH         ;
  SBB A            ; A = 0FFH GDY BIT ROWNY 1
  ORA B            ; SUMA Z WARTOSCIA BIT PORTU 2
  ORA D            ; WYTNIJ BIT PORTU 1
  XRA D
  MOV B, A
  LDA ZMPORT 1     ; STAN PORTU 1
  ANA D
  ORA B            ; WPISZ WYLICZONA WARTOSC
  STA ZMPORT 1
  OUT PORT 1       ; USTAW BIT PORTU 1
```

Drugi przedstawiony w przykładzie sposób, mimo że dłuższy, jest bardziej uniwersalny, gdyż kod procedury nie zależy od pozycji bitów w portach wejściowych i wyjściowych.




Rys. 2. Sposób sterowania linii portu wyjściowego w funkcji stanu linii wejściowych a) przez przesuwanie bitów wejściowych; b) przez wyliczanie wartości logicznej

## PROGRAMY REALIZACJI FUNKCJI CZASOWYCH

W bardziej złożonych systemach do pomiaru czasu jest wykorzystywany specjalizowany układ (programowany zespół liczników). W prostszych zastosowaniach, gdy procesor nie jest zaangażowany w realizację dodatkowych zadań, może on zająć się zliczaniem cykli zegarowych. Znając czas wykonywania poszczególnych instrukcji można napisać procedurę (trwającą przez wymagany czas). Poniżej przedstawiono przykłady programów (pętli czasowych) realizujących żądane opóźnienia.

```
; OPÓŹNIENIE CZASOWE Z 1 REJESTREM
        MVI A, TIM  ; PARAMETR PĘTLI
LOOP:   DCR A
        JNZ LOOP    ; DŁUGOŚĆ = 7 BAJTÓW
                    ; CZAS = 7 + 15 + TIM (10 µs do 1847 µs)

; OPÓŹNIENIE CZASOWE Z 2 REJESTRAMI
        MVI C,TIM1  ; ZEWNĘTRZNY PARAMETR PĘTLI
LOOP1:  MVI A,TIM2  ; WEWNĘTRZNY PARAMETR PĘTLI
LOOP2:  DCR A
        JNZ LOOP2   ; PĘTLA WEWNĘTRZNA
        DCR C       ; PĘTLA ZEWNĘTRZNA
        JNZ LOOP1   ; DŁUGOŚĆ = 12 BAJTÓW
                    ; CZAS = 7 + TIM1* (22 + 15*TIM2)
                    ; 44TC DO 988679 TC (21 µs DO 474 ms)

; OPÓŹNIENIE CZASOWE Z 3 REJESTRAMI
; PRZYKŁAD 1
        LXI B,TIM3  ; PARAMETR PĘTLI
LOOP3:  DCX B       ;
        MOV A, B
        ORA C       ; DŁUGOŚĆ = 9 BAJTÓW
        JNZ LOOP3   ; CZAS = 10 + 24*TIM3
                    ; 34 TC DO 157287 TC (16 µs DO 775 ms)
; PRZYKŁAD 2
        MVI B,TIM4  ; PARAMETR PĘTLI ZEWNĘTRZNEJ
LOOP4:  MVI C,TIM5  ; PARAMETR PĘTLI ŚRODKOWEJ
LOOP5:  MVI A,TIM6  ; PARAMETR PĘTLI WEWNĘTRZNEJ
LOOP6:  DCR A
        JNZ LOOP6   ; PĘTLA WEWNĘTRZNA
        DCR C
        JNZ LOOP5   ; PĘTLA ŚRODKOWA
        DCR B
        JNZ LOOP4   ; PĘTLA ZEWNĘTRZNA
                    ; DŁUGOŚĆ = 18 BAJTÓW
                    ; CZAS = 7 + TIM4* (22 + TIM5* (22 +
                      + 15*TIM6))
                    ; 66 TC DO 253 105664 TC (32 µs do 121 sek)
```

Wykorzystując programową metodę realizacji opóźnienia czasowego należy pamiętać o dodatkowych opóźnieniach wnoszonych przez pozostałe ścieżki programu, jeżeli zależy nam na dokładnym odmierzeniu czasu.

## PROGRAMY PRZETWARZANIA KODÓW

Poniżej przedstawiono programy zmiany kodów BCD na binarny i odwrotnie. Z uwagi na prostsze operacje arytmetyczne na liczbach binarnych, a jednocześnie bardziej komunikatywne — dla operatora — posługiwanie się liczbami dziesiętnymi, operacja zamiany z kodu BCD na binarny odbywa się po wprowadzeniu danych, a przejście z kodu binarnego na BCD przed wyprowadzeniem danych.

```
; KONWERSJA 4 CYFR BCD W REJ. BC NA 16-BITOWĄ LICZ-
; BĘ BINARNĄ W REJ. HL
BCDHEX: MVI A,4        ; LICZNIK CYFR BCD
        LXI H,0        ; WARTOŚĆ POCZĄTKOWA LICZBY
                         BINARNEJ
BCDH1:  PUSH PSW       ; SCHOWAJ LICZNIK CYFR
        MOV E,L        ; WARTOŚĆ POŚREDNIA WYNIKU
        MOV D,H        ;
        DAD H          ; RAZY 2
        DAD H          ; RAZY 4
        DAD H          ; RAZY 5
        DAD H          ; RAZY 10
        LXI D,400H     ; LICZNIK BITÓW KOLEJNEJ CYFRY
BCDH2:  MOV A,C
        RAL            ; PRZESUŃ LICZBĘ BCD O 1 BIT
        MOV C,A
        MOV A,B
        RAL
        MOV B,A
        MOV A,E
        RAL            ; WPROWADŹ BIT DO REJ. E
        MOV E,A
        DCR D          ; WSZYSTKIE 4 BITY
        JNZ BCDH2
        DAD D          ; DODAJ BCD DO HEX
        POP PSW        ; LICZNIK CYFR BCD
        DCR A
        JNZ BCDH1
        RET

                       ; DŁUGOŚĆ = 35 BAJTÓW
                       ; CZAS WYKONANIA = 0,75 ms

; KONWERSJA 16-BITOWEJ LICZBY BINARNEJ Z REJ. HL
; NA 4 CYFRY BCD W REJ. DE
BINBCD: XRA A          ; WARTOŚĆ POCZĄTKOWA
        LXI B,0FC18H   ; —1000
        CALL ILORAZ
        CPI 0AH        ; LICZBA TYSIĘCY WIĘKSZA
        JNC ERROR      ; OD 10 ?
        LXI B,0FF9CH   ; —100
        CALL ILORAZ
        MOV D,A        ; LICZBA TYSIĘCY I SETEK
        MVI C,0F6H     ; —10
        CALL ILORAZ
        MVI C,0FFH     ; —1
        CALL ILORAZ
        MOV E,A        ; LICZBA JEDNOSTEK I DZIESIĄTEK
        RET
;
ILORAZ: ADD A          ; BIT W LEWO
        ADD A          ; 2 BITY W LEWO
        ADD A          ; 3 BITY W LEWO
        ADD A          ; 4 BITY W LEWO
ILOR1:  PUSH H         ; ZAPAMIĘTAJ RESZTĘ
        DAD B          ; ODEJMIJ JEDNOSTKĘ
        JNC ILOR2
        XTHL           ; MIEŚCI SIĘ
        POP H          ; RÓŻNICA
        INR A          ; WYNIK CZĘŚCIOWY
        JMP ILOR1
ILOR2:  POP H          ; POBIERZ RÓŻNICĘ
        RET

        ; DŁUGOŚĆ = 48 BAJTÓW
        ; CZAS WYKONYWANIA OD 0,5 ms
          DO 2,5 ms
```

Program konwersji liczby z postaci binarnej na BCD sprawdza, czy zamieniana liczba mieści się w czterech pozycjach kodu BCD.

**ADAM PLUTA**
Warszawa

---

**Czytelnicy, którzy zapoznali się z możliwościami mikroprocesora 6809, zapewne zadają sobie pytanie: czy dostępne jest dla niego gotowe oprogramowanie? Z oprogramowaniem układów firmy MOTOROLA nie jest u nas w kraju najlepiej. Nie wynika to jednak z faktu, że oprogramowania tego w ogóle nie ma. Stosunkowo niewielka popularność mikroprocesorów z serii 6800 (trudniejsze do zdobycia niż np. Z80) ogranicza obieg tego oprogramowania do wąskiego kręgu zainteresowanych.**

**W poniższym tekście przedstawiamy dość interesujący pakiet programowy, który upraszcza realizację stosunkowo szerokiej klasy zagadnień. Faktycznie, dysponując tego typu zapleczem, jak układ 6839, jest znacznie łatwiej podjąć decyzję o zastosowaniu w planowanej konstrukcji mikroprocesora 6809.**

## Norma IEEE „w pigułce"

Od pewnego czasu firmy produkujące układy scalone starają się wyjść naprzeciw potrzebom użytkowników poszukujących taniego, standardowego oprogramowania. Stwierdzono, że przy stale rosnących kosztach oprogramowania jedyną drogą obniżenia cen jest masowość produkcji. Recepta bez wątpienia prosta: należy sprzedawać standardowe oprogramowanie w postaci pamięci stałej, programowanej u producenta. Przy masowej produkcji koszty takiego oprogramowania będą bliskie kosztom wytwarzania układu scalonego. Tak postąpiono z systemem operacyjnym CP/M integrując go w jednym układzie z mikroprocesorem Z80. Ostatnio firma MOTOROLA wypuściła na rynek układ MC6839 — pamięć stałą o pojemności 8 K × 8, w której zawarta jest pełna implementacja normy IEEE dotyczącej operacji zmiennoprzecinkowej. Oprogramowanie to przeznaczone jest dla procesora MC6809.

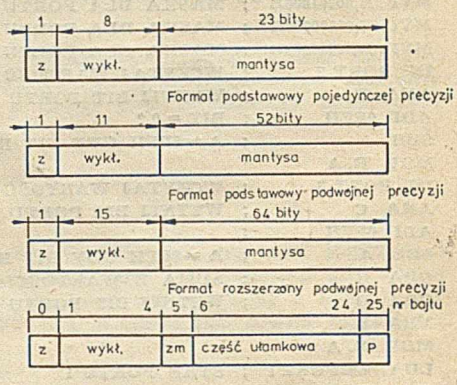
Dość szczegółowy opis tej normy znajdzie Czytelnik w numerze 1'1985 INFORMATYKI. My zatrzymamy się na sposobie, w jaki wskazania normy zostały spełnione w układzie 6839 oraz na przyjętej metodzie komunikacji programu użytkownika z pakietem zmiennoprzecinkowym.

Układ 6839 ma stanowić alternatywę dla rozwiązania sprzętowego w postaci koprocesora lub układu specjalizowanego, np. Am 9511. Oczywiście rozwiązanie oferowane przez 6839 jest o kilka rzędów wolniejsze, ale — również proporcjonalnie tańsze. Projektanci, pragnąc dodatkowo zrekompensować użytkownikowi długi czas obliczeń, wprowadzili — w porównaniu do normy — pewne rozszerzenia. Dotyczą one przede wszystkim diagnostyki błędów i liczby realizowanych operacji.

Program zawarty w układzie napisany jest w sposób przemieszczalny, tzn. działa niezależnie od położenia układu przestrzeni adresowej pamięci procesora. Również „notatnik" (obszar roboczy) w pamięci RAM może być dowolnie umieszczony. Program wykorzystuje 185 komórek na stosie systemowym.

Zgodnie z normą, zmiennoprzecinkowe argumenty operacji występować mogą w następujących formatach (rys. 1):

- podstawowym pojedynczej precyzji
- podstawowym podwójnej precyzji
- rozszerzonym podwójnej precyzji
- rozpakowanym BCD.




**Rys. 1. Formaty zmiennoprzecinkowe**
**z — znak wykładnika, zm — znak części ułamkowej, p — pozycja kropki dziesiętnej**

Ponadto występują formaty stałoprzecinkowe pojedynczej i podwójnej precyzji. Stanowią one, obok formatu

BCD, rozszerzenie w stosunku do ustaleń normy.

Wybór sposobu zaokrąglania wyników zależy od parametrów przekazywanych do pakietu. Przewidziano następujące sposoby zaokrąglania:

- do liczby najbliższej
- w kierunku zera (jest to zgodne z wymogami implementacji FORTRANU)
- w kierunku $+\infty$
- w kierunku $-\infty$

Spełniony jest postulat mówiący o tym, aby w czasie obliczeń zaokrąglanie wystąpiło tylko raz. Nieskończoność może występować w postaci afinicznej bądź rzutowej. Poza tym istnieje specjalny wzorzec dla nieliczb (tzn. takich ciągów bitów, które nie mogą zostać potraktowane jako liczba w żadnym z formatów). Istnieją też określone sposoby przedstawiania liczb nieznormalizowanych i zera.

Procedury rozpoznaja siedem sytuacji wyjątkowych. Każda z nich może spowodować przerwanie programowe bądź przyjąć założony z góry wynik i kontynuować obliczenia. W przypadku przerwania sterowanie przekazywane jest do programu użytkownika, pod uprzednio podany adres.

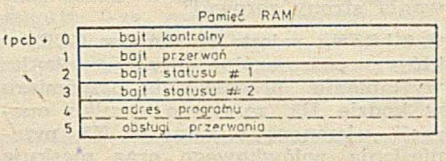
Program zawarty w 6839 rozpoznaje następujące sytuacje wyjątkowe: nielegalna operacja, niedomiar, nadmiar, dzielenie przez zero, niedokładny wynik, nadmiar stałoprzecinkowy, porównywanie liczby z nieliczbą. Przy przerwaniu pakiet dostarcza użytkownikowi informacji o przyczynie przerwania, adresie w programie użytkownika, z którego nastąpił skok do pakietu, argumentach operacji, kodzie operacji zmiennoprzecinkowej oraz proponowanym wyniku zastępczym. Dane te przekazywane są na stosie.

Operacje wykonywane przez program w 6839 mogą być jedno- lub dwuargumentowe. Są to:

- dodawanie, odejmowanie, mnożenie i dzielenie
- wyznaczanie reszty z dzielenia
- różne typy porównań
- pierwiastek kwadratowy
- wyznaczanie części całkowitej liczby
- konwersje formatów
- moduł liczby
- zamiana znaku
- przesłanie wraz z konwersją formatu.

Operacje porównania nie wymagają, aby liczby miały ten sam format.

Istnieją dwa sposoby komunikowania się programu użytkownika z pakietem zmiennoprzecinkowym. Pierwszy — rejestrowy — polega na przekazywaniu adresów argumentów w rejestrach. Drugi — stosowy — przekazuje argumenty na stosie. Obok argumentów do programu zawartego w 6839 należy przesłać adres obszaru roboczego w pamięci RAM, tzw. fpcb. Jego postać przedstawiona jest na rysunku 2.




**Rys. 2. Notatnik w pamięci RAM**

W bajcie kontrolnym muszą znaleźć się dane dotyczące formatu argumentów, sposobu traktowania $\infty$, sposobu zaokrąglania i normalizowania. Bajt przerwań stanowi maskę dla wywoływania przerwań przez określone sytuacje. Dwa bajty statusu zapisywane są przez program 6839 i stanowią opis toku operacji. W przypadku przerwania sterowanie przekazywane jest pod adres zawarty w komórkach fpcb+4 i fpcb+5. Wejścia do pakietu znajdują się w następujących miejscach: adres początkowy MC6839 + $3D_H$ dla komunikacji rejestrowej i dwie komórki dalej — dla stosowej. Za rozkazem skoku do podprogramu zmiennoprzecinkowego powinien znaleźć się bajt będący kodem operacji. Dla przykładu, wywołania procedury mnożenia mogą wyglądać jak następuje:

1) rejestr U := adres arg. 1
   rejestr Y := adres arg. 2
   rejestr D := adres fpcb
   rejestr X := adres wyniku
   LBSR rom + 3D ; „daleki" skok do podprogramu
   ⟨kod operacji mnożenia⟩
2) argument 1 na stos
   argument 2 na stos
   adres fpcb na stos
   LBSR rom + 3F
   ⟨kod operacji mnożenia⟩
   wynik ze stosu.

Jak widać, uruchomienie operacji zmiennoprzecinkowej nie jest trudne. A jak długo trwają operacje? Zależy to od częstotliwości zegara systemowego i od wielkości argumentów. Przykładowo, dla zegara 2 MHz mnożenie trwa maksymalnie 4,8 *us* (format rozszerzony), a konwersja z tego formatu na format BCD (czyli przygotowanie do wyprowadzenia w postaci ciągu kodów ASCII) — maksymalnie 85 *us*. W większości przypadków są to wielkości do przyjęcia.

Kończąc — pozostaje życzyć sobie i wszystkim Czytelnikom mikroKLANU, aby podobne programy, nawet niekoniecznie w postaci pamięci ROM 8 K × 8, produkowanej przez CEMI (sic!), pojawiły się dla procesorów, na których opiera się rodzimy konstruktor. Autor jest jednak pod tym względem sceptyczny.

**JERZY ORKISZEWSKI**
Warszawa

---

**Po tegorocznych Targach Poznańskich oczy konstruktorów sprzętu zwrócone są na mikroprocesor 16-bitowy 8086. Okazało się bowiem, że firmy polonijne potrafiły w przeciągu zaledwie kilku miesięcy przekopiować zachodnie rozwiązania, mimo ich stosunkowo dużego stopnia zaawansowania technicznego.**

**O wadach i zaletach mikroprocesora 8086 już niebawem będzie można przeczytać na łamach mikroKLANU. Zanim to jednak nastąpi, chcemy zwrócić uwagę Czytelników na inny mikroprocesor. Chodzi o to, by nie przechodzić bezkrytycznie na technikę 16-bitową tam, gdzie nie jest to w pełni uzasadnione. Konstrukcje 16-bitowe są znacznie droższe i to nie tylko ze względu na cenę samego mikroprocesora. Tymczasem w wielu zastosowaniach zwykła „ósemka" z powodzeniem da sobie radę. Trzeba tylko pamiętać, że świat mikroprocesorów 8-bitowych nie kończy się na 8080.**

**Mikroprocesor 6809 cechuje stosunkowo duża moc przetwarzania, może więc być użyteczny np. w sterowniku obciążonym zagadnieniami optymalizacji lub złożonych algorytmach kompensacyjnych itp.**

**Nie polecamy natomiast 6809 do budowy komputerów osobistych. Nie dlatego, że nie nadaje się on do tego typu zastosowań. Po prostu dla komputerów pracujących pod kontrolą systemu operacyjnego CP/M lub kompatybilnych z IBM PC istnieje już znaczna liczba programów użytkowych, których nie opłaca się pisać jeszcze raz.**

## MC6809 — w pół kroku między 6800 a 68000

Procesor 6809 wart jest przedstawienia z kilku powodów. Po pierwsze, stanowi wersję rozwojową popularnego 6800, prekursora rodziny. Po drugie, pomimo 8-bitowej szyny danych ma rozbudowaną grupę rozkazów operujących na danych 16-bitowych, co czyni zeń procesor „quasi 16-bitowy". Po trzecie, i to jest właśnie jego cechą najważniejsza, ma bardzo rozbudowane tryby adresowania oraz rozkazy i mechanizmy spotykane właściwie dopiero w mikroprocesorach 16-bitowych. Dostępne rozkazy pozwalają na pisanie programów zwięzłych i nowoczesnych; w szczególności przemieszczalnych (ang. position-independent), współużywalnych i o modularnej budowie. Znacznie ułatwione jest tworzenie programów działających w czasie rzeczywistym oraz oprogramowania systemowego, w tym kompilatorów języków wysokiego poziomu.

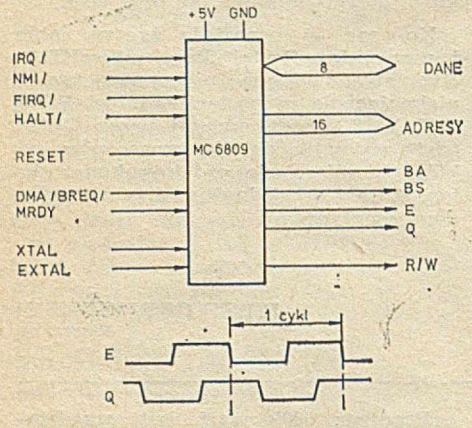
Procesor kosztuje obecnie ok. 18 marek zachodnioniemieckich, a więc

jest kilkakrotnie tańszy niż 8088 czy 68008.

Mikroprocesor MC 6809 pojawił się w 1978 r. Pierwotnie jego perspektywy rynkowe, wobec istniejących już wtedy mikroprocesorów 16-bitowych, oceniane były dość sceptycznie. Rzeczywistość okazała się jednak wcale nie taka czarna. Obecnie jest on produkowany przez kilka firm (z europejskich — EFCIS), a jego sprzedaż osiąga 3 mln sztuk rocznie.

**Szyny i sygnały sterujące**

Wyprowadzenia procesora przedstawione są schematycznie na rysunku. Procesor ma 16-bitową szynę adresową i 8-bitową szynę danych. Wbudowany generator wymaga jedynie uzupełnienia kwarcem o częstotliwości 4-krotnie większej niż częstotliwość szyny. O kierunku przesłania danych decyduje linia R/—W. Linie E i Q to sygnały zegarowe. W stosunku do 6800 nowością jest sygnał Q (rys.). Układ ma trzy wejścia przerwań sprzętowych. Obok typowych linii IRQ/ i NMI/ występuje linia FIRQ/. Wejście to ma wyższy priorytet niż IRQ/, a przyjęcie tego przerwania poprzedza przesłanie na stos wyłącznie licznika rozkazów i rejestru warunków. Przyspiesza to znacznie obsługę przerwania.




**Wyprowadzenia MC6809 i przebiegi zegarowe**

Stan procesora wskazywany jest przez poziomy na liniach BA i BS. Różnicowane są następujące stany: praca normalna, cykl przyjęcia przerwania, oczekiwanie na zewnętrzny sygnał synchronizujący oraz potwierdzenia zwolnienia szyn.

Szyny udostępniane są w wyniku rozpoznania niskiego poziomu na linii DMA/-BUS REQ/. Linia MRDY służy do współpracy z wolnymi pamięciami. Procesor zerowany jest sygnałem RESET/.

W porównaniu do 6800 wyeliminowano linię Valid Memory Address. Stało się to możliwe, ponieważ wszystkie cykle szyny 6809 są „ważne". Jeżeli procesor nie odwołuje się w danym cyklu do pamięci, to ustawia linie A0—A15 oraz R/—W w stan wysoki, nie dokonując jednak odczytu. Sygnał Q może służyć do synchronizacji w systemie, nie jest jednak niezbędny do sterowania układów peryferyjnych rodziny M6800.

**Rejestry procesora**

Programista ma dostęp do ośmiu rejestrów. Są to:

- dwa 8-bitowe akumulatory oznaczane jako A i B. Przy wykonywaniu rozkazów 16-bitowych oba te rejestry tworzą jeden 16-bitowy akumulator D
- 16-bitowy licznik rozkazów
- cztery 16-bitowe rejestry indeksowe X, Y, S i U. Dwa z tych rejestrów są również wskaźnikami stosów: systemowego rejestru (S) i użytkownika rejestru (U)
- 8-bitowy rejestr strony zerowej DP. Jego zawartość tworzy bardziej znaczący bajt adresu w trybie „adresowania strony zerowej"
- 8-bitowy rejestr warunków CCR, w którym poszczególne bity sygnalizują: wystąpienie przeniesienia, nadmiaru w kodzie U2, zerowego wyniku operacji, ujemnego wyniku w U2, przeniesienia połówkowego oraz maskują przerwania IRQ/ i FIRQ/. Ostatni bit wskazuje, czy ostatnio przyjęte przerwanie było zwykłe czy „szybkie" (FIRQ/).

Na podkreślenie zasługuje istnienie dwu odrębnych stosów, co znacznie ułatwia przekazywanie parametrów do podprogramów i wykonywanie obliczeń numerycznych. Aby zapewnić zgodność programową z 6800 rejestr DP jest zerowany sygnałem RESET/.

**Tryby adresowania**

Tryby adresowania są najsilniejszą stroną omawianego procesora. Podana niżej ich lista obejmuje również przykładowe rozkazy.

**Tryb bezpośredni** (ang. inherent). Dotyczy on operacji na rejestrach. Przykładem jest rozkaz MUL — pomnóż zawartość A przez zawartość B i wynik umieść w D.

**Tryb natychmiastowy** (ang. immediate). Dana znajduje się zaraz za kodem rozkazu. Np. LDD # $8000 — załaduj akumulator D stałą $8000_H$. W tym trybie mieszczą się również rozkazy przesłań międzyrejestrowych TFR i wymiany zawartości rejestrów EXG. O tym, które rejestry biorą udział w operacji, decyduje bajt kodowy umieszczony bezpośrednio za kodem rozkazu (ang. postbyte).

**Tryb rozszerzony** (ang. extended). W tym trybie adres efektywny argumentu podany jest explicite po kodzie rozkazu; np. LDD 5403 — załaduj akumulator D dana 16-bitowa z komórek pamięci o adresach $5403_H/5404_H$. Ten tryb adresowania nie jest preferowany z uwagi na to, że wyklucza przemieszczalność programu.

**Tryb strony zerowej** (ang. direct). Adres efektywny argumentu tworzony jest przez złożenie zawartości rejestru DP i bajtu usytuowanego za kodem rozkazu. Dla przykładu, jeżeli rejestr DP zawiera $80_H$, to wykonanie rozkazu LDA > 3F spowoduje załadowanie akumulatora A daną z komórki pamięci o adresie $803F_H$.

**Tryb indeksowy** (ang. indexed). Tryb ten ma wiele odmian. Są to:

A) Przesunięcie stałe — 5-, 8- lub 16-bitowe. Przesunięcie, w kodzie U2, dodawane jest do zawartości jednego z rejestrów indeksowych: np. LDY--32798, U — załaduj 16-bitowy rejestr Y daną umieszczoną o 32 798 komórek poniżej adresu znajdującego się w rejestrze U. Inny przykład, to CMPD 279, PCR — porównaj zawartość akumulatora D z zawartością dwu komórek pamięci, pod adresem o 279 komórek większym niż aktualna zawartość licznika rozkazów (licznik rozkazów może służyć jako dodatkowy rejestr indeksowy).

B) Przesunięcie znajduje się w jednym z rejestrów 8- lub 16-bitowych. Rozkaz LDX D,U spowoduje załadowanie rejestru X daną z komórek pamięci wyznaczonych przez dodanie zawartości D i U.

C) Przesunięcie równe jest zero — zawartość rejestru indeksowego ulega automatycznie zmianie o 1 bądź 2. Tak działają rozkazy LDB,X+ — pobierz do akumulatora B dana wskazywaną przez rejestr X, a następnie zwiększ X o 1 oraz LDB,--S — zmniejsz zawartość rejestru S o 2, a następnie załaduj dana według nowego adresu. Wprowadzono również dodatkowy poziom wyznaczania adresu (ang. indirection). Dotyczy to trybów indeksowego i rozszerzonego. W szczególności procesor wykonuje rozkazy typu: LDA [F400] — pobierz do A dana, której adres znajduje się w komórce o adresie $F400_H$; JMP [30,PCR] — skocz do adresu, który znajduje się w komórkach wyznaczonych przez sumę aktualnej zawartości licznika rozkazów i stałej $30_H$ czy LDD [,Y++] — pobierz do akumulatora D dana, której adres znajduje się w komórkach wskazywanych przez Y, a następnie zwiększ Y o 2.

Sterowanie w programie realizują „bliskie" i „dalekie" skoki względne (ang. branch relativ). W przypadku spełnienia warunku i wykonania skoku sterowanie przekazywane jest do adresu będącego sumą zawartości licznika rozkazów i stałej w kodzie U2, która umieszczona jest za kodem rozkazu. Rozkazy skoków bliskich (stała 8-bitowa) umożliwiają skoki na odległość —128 ... +127 komórek, a skoki dalekie (stała 16-bitowa) —32768 ... +32767.

**Lista rozkazów**

Procesor wykonuje 59 typów rozkazów, biorąc jednak pod uwagę wszystkie możliwe warianty adresowania,

uzyskuje się ogromną liczbę 1464 dostępnych operacji. Rozkazy kodowane są na 1, 2, 3, 4 bądź 5 bajtach. Liczba cykli maszynowych, w czasie których wykonywane są rozkazy, wynosi dwa do ośmiu (wyjątkowo tylko 20, w przypadku przerwania programowego). Przy częstotliwości zegara systemu 2 MHz dodawanie 16-bitowe trwa 2 $\mu$s, a mnożenie danych 8-bitowych 5 $\mu$s. Procesor nie dysponuje odrębnymi rozkazami wejścia/wyjścia.

Rozkazy, dotyczące akumulatorów 8-bitowych i pojedynczych komórek pamięci, obejmują operacje: dodawania, odejmowania, testowania bitów i słów, inkrementacji i dekrementacji, przesunięć i obrotów, operacje logiczne AND, OR i EXOR, mnożenie (!), zerowanie, uzupełnianie do 1 i 2, korekcję dziesiętną.

Operacje 16-bitowe, to: dodawanie, odejmowanie, porównanie, przesłania oraz przedłużenie znaku — czyli zamiana danej 8-bitowej w kodzie U2 na daną 16-bitową w tym kodzie.

Bogaty jest zestaw rozkazów operujących na stosie i działających na rejestrach indeksowych. Rozkazy przesłań na stosy użytkownika — PSHU i systemowy — PSHS przesyłają dowolne rejestry pojedynczo lub grupowo. Dotyczy to również operacji zdjęcia ze stosu — PULU i PULS. Wskaźniki stosu mogą być przesyłane (do) i z pamięci oraz brać udział w operacjach porównania. Istnieją również rozkazy obliczania adresu efektywnego — LEA (ang. Load Effective Address). Za ich pomocą można do jednego z rejestrów indeksowych załadować adres danej bez wykonywania na niej operacji. Na przykład LEAY —10,S ładuje do rejestru Y adres utworzony przez odjęcie od zawartości S stałej 10, LEAS 8,S wykonuje operację S:=S+8 (czyli „oczyszcza" 8 komórek stosu), a LEAU —D,U rezerwuje na stosie użytkownika tyle komórek, ile wynosi zawartość akumulatora D.

Rozkazy skoków względnych mogą testować bity CCR, bądź relacje zachodzące w wyniku wykonywania operacji na danych w kodach U2 lub NKB. Wszystkie skoki występują w formie „bliskiej" i „dalekiej". Ponadto istnieją rozkazy skoków do podprogramów i skoku bezwarunkowego.

Ostatnią grupę stanowią tzw. rozkazy różne. Wśród nich na uwagę zasługują trzy rozkazy przerwań programowych SWI, SWI2 i SWI3 oraz rozkaz SYNC, który pozwala na synchronizację programu ze zdarzeniami zewnętrznymi. Powoduje on wstrzymanie wykonywania programu do chwili przyjścia przerwania na linii IRQ/. Jeżeli jednak przerwanie to jest zablokowane, to program wykonywany jest dalej.

* * *

Mikroprocesor MC6809 jest produkowany w dwu wersjach. Obok 6809 występuje 6809E przeznaczony do pracy wieloprocesorowej. Dostępne są też układy towarzyszące, znacznie rozszerzające możliwości procesora. Dla przykładu układ zarządzania pamięcią MC6829 zwiększa obszar adresowy do 2Mb, a układ MC6883 spełnia rolę zegara systemu zintegrowanego ze sterownikiem pamięci dynamicznej, w tym układów 4164.

Możliwości programowe, jakie znajdujemy w mikroprocesorze 6809 nie są odległe od tych, które oferują procesory „w pełni 16-bitowe". Powyższy przegląd daje nam zatem obraz tego, czym będziemy się zajmować w przyszłości. Jak odległej, to już inne zagadnienie.

**JERZY ORKISZEWSKI**
Warszawa

---

**Po olbrzymim zainteresowaniu wydrukiem kodu maszynowego programu fig-FORTH spodziewaliśmy się zalewu publikacjami nadsyłanymi przez zwolenników tego języka. Jak zwykle okazało się, że wielu chętnie korzysta, natomiast niewielu chce przekazać coś innym. Marek Czarzyński po nadesłaniu odcinka, który zamieściliśmy w 13 mikroKLANIE, i deklaracji wznowienia współpracy ponownie zamilkł. Część winy spada jednak na opóźnienie, z jakim w bieżącym roku ukazuje się INFORMATYKA. W chwili gdy oddajemy do drukarni ten numer (połowa czerwca), do rąk Czytelników trafił dopiero pierwszy numer czasopisma. Tak więc zwolennicy FORTHA nic jeszcze nie wiedzą o „kryzysie kącika" — co częściowo tłumaczy ich bierność. Ciszę przerwał Michał Choroszucha, który próbował wykorzystać procedury zamieszczone w 13 mikroKLANIE i... natknął się na wiele błędów. W poniższym materiale próbuje je sprostować, reklamując równocześnie wykonaną przez siebie modyfikację programu fig-FORTH.**

## Wokół języka FORTH

Poniższy tekst jest efektem analizy teoretycznej, a następnie próby uruchomienia procedur opisanych przez Marka Czarzyńskiego w 13 numerze mikroKLANU (INFORMATYKA, nr 1, 1985). Procedury były uruchamiane za pomocą zmodyfikowanej wersji programu 8080 fig-FORTH 1.1 (wersja 1.1 M), współpracującej z plikami systemu operacyjnego CP/M zamiast (fizycznie) z dyskietką, jak to występuje w wersji oryginalnej. Modyfikacja ta stanowi część pracy dyplomowej autora. Wersje źródłowe programów 8080 fig-FORTH 1.1 oraz 1.1 M są dostępne u autora [1]).

Przed opisem prawidłowych (działających) procedur przedstawione zostaną niektóre zmiany występujące w wersji 1.1 M — nie mające bezpośredniego związku z operacjami na plikach systemu CP/M. Zmodyfikowana procedura **ID.** zeruje najbardziej znaczący bit przy drukowaniu ostatniego znaku nazwy. **CREATE** „zachowuje się" prawidłowo także przy nazwach dłuższych od wartości zmiennej **WIDTH** — w przeciwieństwie do procedury oryginalnej. **VLIST** nigdy nie przekracza 79 znaków w linii — niezależnie od długości drukowanych słów. Jedynym wyjątkiem jest tu wydruk tekstu OK przez słowo **QUIT** po zakończeniu działania **VLIST.**

**Nowe procedury:**

**?NUMBER** (——d) czyta z klawiatury ciąg cyfr zakończonych znakiem powrotu karetki (CR). Wynik: w postaci liczby podwójnej na stosie parametrów oraz odpowiedniej wartości zmiennej **DPL.** Procedura wykorzystywana jest do czytania parametrów

```
: ID.
   PAD   [ HEX ]   BL   5F   FILL
   DUP   PFA   LFA   OVER   -   PAD
   SWAP   <MOVE   PAD   COUNT   1F   AND
   ( FOLLOWING APPENDED )   2DUP   +   1-
   80   TOGGLE   ( END OF MODIFICATIONS )
   TYPE   SPACE   ;

: CREATE
   -FIND   [ HEX ]
      IF   ( FOUND )
         DROP   NFA   ID.   4   MESSAGE   SPACE
      ENDIF
   HERE   DUP   C@   WIDTH   @   MIN
   ( FOLLOWING APPENDED )   2DUP   OAO
   OR   SWAP   C!   ( END OF MODIFICATIONS )
   1+   ALLOT   ( DELETED : DUP OAO TOGGLE )
   HERE   1-   80   TOGGLE   LATEST   ,
   CURRENT   @   !   HERE   2+   ,   ;

: VLIST
   [ HEX ]   80   OUT   !   CONTEXT   @   @
      BEGIN
         DUP   @   1F   AND   OUT   @   +   4C   >
            IF   ( LINE TOO LONG )
               CR
            ELSE
               SPACE   SPACE
            ENDIF
         DUP   ID.   PFA   LFA   @
         DUP   0=   ?TERMINAL   OR
      UNTIL   ( REPEAT UNTIL TRUE )
   SPACE   SPACE   DROP   ;

: ?NUMBER
   HERE   [ HEX ]   20   EXPECT
   HERE   #L   ERELOS   DROP
   HERE   +   EL   SWAP   C!
   +   1-   NUMBER   ;

: ?SPACES
   OUT   @   -   SPACES   ;

: TAB
   >R   OUT   @   0
      BEGIN
         2DUP   <   0=   ( )=   >
      WHILE   ( EXIT IF TRUE )
         R   +
      REPEAT   ( UNTIL TRUE DURING WHILE )
   R>   DROP   SWAP   -   SPACES   ;

: CCR
   CR   0   OUT   !   ;

: 1-
   1   -   ;

: 2-
   2   -   ;
```

Wydruk 1

[1]) Listy nadesłane do redakcji przekażemy autorowi (przyp. red.).

przez program w trybie interakcyjnym.
**?SPACES** (n ———) ustawia kursor w kolumnie „n"; jeśli jest to niemożliwe, to pozycja kursora pozostaje bez zmiany. Duże dodatnie wartości „n" nie są eliminowane.
**TAB** (n ———) tabulacja co „n' kolumn
**CR** w wersji 1.1 M spełnia funkcję procedury **CCR.**

**Opis błędów w procedurach z nr 13 mikroKLANU**

**FIND** — w przypadku nieobecności poszukiwanego słowa w słowniku nie przerywa działania programu, co powoduje pobranie adresu z pustego stosu i wydruku niewłaściwych tekstów.
**NFA-FOR-CFA** — brakuje powielenia adresu **DUP**
(błąd MSG # 1 — pusty stos). Można też zastosować jedną z procedur alternatywnych.

```
: FIND   ( CHANGED )
  -FIND
    IF
      DROP  2-
    ELSE
      0  ERROR
    ENDIF  ;

: NFA-FOR-CFA   ( CHANGED )
  3  -
    BEGIN
      1-  DUP  C@  [ HEX ]  80  )
    UNTIL  ;

: NFA-FOR-CFA'   ( ALTERNATIVES FOR  NFA-FOR-CFA )
  2+   ( PFA )   NFA
  ( OR : 3  -  -1   TRAVERSE )  ;

: NAME?   ( CHANGED ,  *WARNING* )
  ( POSSIBLE SYSTEM ERROR IF ADDRESS OUT OF MEMORY )
  DUP   NFA-FOR-CFA   DUP   C@
  1F   AND   +   3   +   =  ;

: NAME(PFA)
  2-   ( MINUS NOT TYPED IN SOURCE LISTING )
  NAME(CFA)  ;

: L2
  DUP   0   OVER   ." ADR="   .
  ( TABULATION IMPROVED )
  [ HEX ]   0C   ?SPACES   ." HEX="   DUP   .
  17   ?SPACES   DUP   ." NAME="   NAME?
    IF
      NAME(CFA)
    ELSE
      ." ????"   DROP
    ENDIF  ;
```

```
: ;S?   ( *WARNING* )
  ( UNDEFINED REPLY IF WORD DOESN'T END WITH  'SEMIS' )
  2+   DUP   @   2+   '   ;S   =  ;

: E."   ( MODIFIED ,  DELETED :  E00FF ,  EFF00 )
  1+   )R   R   +   R)
    DO   I   C@   EMIT   LOOP  ;
```

**Wydruk 2**

**NAME?** — występują błędy przy próbie wydruku słowa, w którym jest kompilator, np.:
: END [COMPILE] UNTIL ; IMMEDIATE
**NAME(PFA)** — brakuje znaku „—"
**L2** — formowanie wydruku nie działa tak, jak to zostało opisane w „spodziewanym wydruku" (strona 19).
(.")? — brakuje znaku „?" w nazwie.
"? — można zastąpić ʙ FF AND przez C ʙ
**E00FF, EFF00, E."** — można znacznie uprościć. (Wydruk 2)
W procedurach **L1** oraz **LOOK-IN** należy zastąpić **CR** przez **CCR.** Pozostałe procedury nie wymagają zmian.

★ ★ ★

Stosunkowo rzadko używanym, a bardzo pożytecznym słowem FORTHA jest VOCABULARY. Przykłady jego zastosowania znalazły się w 10 mikroKLANIE (INFORMATYKA, nr 10, 1984). Niestety w opublikowanej na stronie 20 sekwencji słów znalazł się poważny błąd, który pozwala przypuszczać, że opublikowane programy nie były uruchamiane przez autora artykułu. Jak wspomniano w cytowanym artykule, stosowanie wielu słowników umożliwia pisanie programów bez konieczności rezerwowania dla nich unikalnych nazw procedur. Każda próba definiowania już istniejącej procedury powoduje ostrzeżenie: ERROR # 4: NAME NOT UNIQUE, natomiast próba kompilacji do nowej procedury słowa nie występującego w danym słowniku (ani poniżej niego, aż do końca słownika FORTH włącznie) kończy się błędem: ERROR # 0: INPUT CONVERSION. Próby ominięcia zabezpieczeń FORTHA przez zmianę wartości zmiennej CONTEXT mijają się z celem (wydruk 1), czego przyczynę łatwo znaleźć analizując działanie kompilatora :. Poniższa definicja została napisana w oparciu o [1]:

```
: : ?EXEC !CSP CURRENT ʙ CONTEXT !
  CREATE ] ;CODE IMMEDIATE [HEX]
  128 (RPP) LHLD H DCX B M MOV H DCX
  C M MOV 128 SHLD D INX E C MOV D B MOV
  NEXT (145)     JMP C;
```

Aby ominąć zabezpieczenia FORTHA zawarte w słowie ; należy odpowiednio ustawić wartość zmiennej CONTEXT dopiero po utworzeniu nagłówka definicji (wydruk 1). Przywrócenie stanu początkowego CONTEXT po zakończeniu definiowania słowa nie jest konieczne.

```
8080 FIG-FORTH 1.1 M 24K CP/M
READY
FORTH DEFINITIONS OK
: F1 ; VOCABULARY VOC-A : F3 ; VOCABULARY VOC-B OK
VOC-B DEFINITIONS OK
: B1 ; : B2 ; VOCABULARY VOC-B.B : B4 ; OK
FORTH DEFINITIONS OK
: F5 ; VOCABULARY VOC-C OK
VOC-C DEFINITIONS OK
FORTH VOC-B  ( OR :  VOC-B  ONLY ) OK
: C1 B1 B2 B4 ; B1 ?_MSG ? 0
VLIST
    VOC-C   F5   VOC-B   F3   VOC-A   F1   TASK   OK
: C1          ( SETS CONTEXT TO CURRENT = VOC-C )
[ VOC-B ]     ( SETS CONTEXT TO VOC-B )
B1 B2 B4      ( COMPILES VOC-B WORDS )
[ VOC-C ]     ( RESTORES CONTEXT TO VOC-C )  ; OK
VLIST
C1      VOC-C   F5   VOC-B   F3   VOC-A   F1   TASK   OK
```

**Wydruk 1**

W cytowanym artykule znalazło się stwierdzenie, że zmiana wartości stałej jest niedozwolona. Nie jest to prawda; zalecany sposób zmiany wartości stałej został opisany w [2]. Używa się do tego celu słowa ' (po angielsku nazywa się je: tick) w następujący sposób:
value CONSTANT name
new-value ' name !

```
CR DECIMAL 240 LIST 241 LIST

SRC # 240
  0 ( MEMORY DUMP PRIMATIVES  **)
  1
  2 DECIMAL
  3 : (EMIT)   ( CHAR ---   EMIT DISPLAYABLE CHAR ,  ELSE .  *)
  4    127   AND   DUP   BL   (
  5       IF   DROP   46   ( "." )   ENDIF
  6    EMIT   ;
  7 : (DUMP)   ( FIRST --- NEXT   DUMP LINE  *)
  8    DUP   5   .R   ( .ADDRESS )   SPACE
  9    DUP   16   ( ADDR ADDR 16 )
 10    OVER   +   SWAP   ( ADDR 16+ADDR ADDR )
 11       DO   I   C@   3   .R   LOOP   ( ADDR )
 12    SPACE   SPACE   16   0
 13       DO   ( ADDR )   DUP   C@   (EMIT)   1+   ( ADDR+ )   LOOP
 14    CR   ( 16+ADDR )   ;
 15 --)

SRC # 241
  0 ( MEMORY DUMP WORD  **)
  1
  2 : DUMP   ( FIRST LAST ---   DUMP MEMORY )
  3    BASE   @   )R   HEX   )R   CR   ( FIRST )
  4       BEGIN
  5          (DUMP)   DUP   R   U(   0=   ?TERMINAL   OR
  6       UNTIL   ( TRUE ,  IF CURRENT )= LAST  OR  KEY PRESSED )
  7    R)   DROP   DROP   R)   BASE   !   ;   ;S
  8
OK
240 LOAD OK
VLIST
DUMP   (DUMP)   (EMIT)   TASK   OK
256 512 DUMP
  100   0 C3 85 17  0 C3 70 17  1  1  1  E B0 20 7F  0  .C...Cp.....0 .
  110  30 3F  0 3E 30 3F  0 3E 1F  0  0  0 73 20 BB 20  0?.)0?.)....s ;
  120  64 20  5  0 20 B3 30 3F 30 3F  0  0 21 82 3F 7E  d .. 30?0?..!.?~
  130  B9 C2 3D  1 23 7E B8 C2 3D  1  0  0  0  A  3 6F  9B=.#~8B=......o
  140  C3 48  1 D5 E5  A  3 6F  A  3 67 5E 23 56 EB E9  CH.Ue..o..g^#VkI
  150  83 4C 49 D4  0  0 58  1  A  3 6F  A  3 67 C3 44  .LIT..X...o..gCD
  160   1 87 45 58 45 43 55 54 C5 50  1 6D  1 E1 C3 4B  ..EXECUTEP.m.aCK
  170   1 86 42 52 41 4E 43 C8 61  1 7C  1 60 69 5E 23  ..BRANCHa.|.`i^#
  180  56 2B 19 4D 44 C3 45  1 87 30 42 52 41 4E 43 C8  V+.MDCE..0BRANCH
  190  71  1 94  1 E1 7D B4 CA 7C  1  3  3 C3 45  1 84  q....a}4J|...CE..
  1A0  28 44 4F A9 88  1 A8  1 2A 86 3F 2B 2B 2B 2B 22  (DO)..(.*.?++++"
  1B0  86 3F D1 73 23 72 D1 23 73 23 72 C3 45  1 86 28  .?Qs#rQ#s#rCE..(
  1C0  4C 4F 4F 50 A9 9F  1 C9  1 11  1  0 2A 86 3F 7E  LOOP)..I....*.?~
  1D0  83 77 5F 23 7E 8A 77 23 14 15 57 FA E6  1 7B 96  .w_#~.w#..Wz.{.
  1E0  7A 23 9E C3 EB  1 7E 93 23 7E 9A FA 7C  1 23 22  z#.Ck.~.#~.z|.#"
  1F0  86 3F  3  3 C3 45  1 87 28 2B 4C 4F 4F 50 A9 BE  .?..CE..(+LOOP)
OK
BYE
```

**Wydruk 2**

Poniżej zostaną przedstawione słowa służące do wyprowadzania zawartości pamięci [1]). Drukowane są kody szesnastkowe poszczególnych bajtów pamięci oraz ich odpowiedniki według kodu ASCII. Procedury można łatwo modyfikować dla dostosowania ich do innych długości linii na ekranie, można również drukować słowa pamięci, a nie poszczególne bajty (należy użyć konstrukcji ... 2 +LOOP).

Dla dociekliwych: znak o kodzie HEX 7F (ang. delete) nie jest obrazowany przez drukarkę, co można uwzględnić w słowie ⟨EMIT⟩. Jeśli ktoś chce oglądać obszar pamięci poza pamięcią zajętą przez FORTHA, konkretnie powyżej HEX 8000, to lepiej niech zmieni ósmą linię kadru 240 (wydruk 2) na:

DUP 0 5 D.R SPACE

W przeciwnym wypadku adresy będą podawane jako liczby ujemne.

Długie wydruki można przerwać przez wciśnięcie dowolnego klawisza (radzę stosować znak „delete", gdyż nie daje to żadnych widocznych efektów w następnej linii ekranu). W słowie (DUMP) zaprezentowane zostały dwie spośród trzech najczęściej stosowanych technik programowania pętli DO:

(start count) OVER + SWAP

DO ... I (start...start+count—1) ... LOOP ( )
(start count 0)

DO ... 1+ ... LOOP (start+count)
(start count 0)

DO ... DUP I + (start start+I) ... LOOP (start)

W nawiasach podano stan istotnych liczb na stosie parametrów zgodnie z [1], czyli: wierzchołek stosu po prawej stronie, jeśli występuje ciąg vvv ——— yyy, to znaczy, że vvv jest stosem przed, a yyy — po wykonaniu definicji; brak liczby oznacza stos pusty (w stosunku do sytuacji określonej jako poprzednia).

```
240 LIST
SRC ⊕ 240
  0 ( FORTH79 WORDS :  DEPTH  PICK ;  !HEAP  **)
  1 : DEPTH   ( --- STACK_DEPTH  *)
  2   S0  @  SP@  -  2-  2  /  ;
  3 : (HEAP)   ( i-1 --- ADDR_Ni  *)
  4   DEPTH  2-  DUP  0(  9  ?ERROR
  5   OVER  U(  9  ?ERROR
  6   1+  DUP  +  SP@  +  ;
  7 : PICK   ( i --- Ni  *)
  8   1-  (HEAP)  @  ;
  9 : !HEAP   ( value i ---   MODIFY STACK  *)
 10   DUP  0=  9  ?ERROR
 11   (HEAP)  !  ;
 12 ;S
 13
 14
 15
OK
240 LOAD OK
VLIST
!HEAP  PICK  (HEAP)  DEPTH  TASK  OK
1 3 2 4 5 0 DEPTH . 6 OK
9 4 !HEAP 7 2 !HEAP OK
. . . . . 0 7 4 9 3 1 OK
5 4 8 2 1 3 PICK . . . . . 8 1 2 8 4 5 OK
BYE
```

**Wydruk 3**

Wydruk 3 przedstawia przykładową implementację w języku fig-FORTH 1.1 słów języka FORTH79: **DEPTH** i **PICK** (patrz mikroKLAN 2, INFORMATYKA nr 2, 1984) oraz nowe słowo **!HEAP** służące do modyfikacji stosu względem jego wierzchołka.

W następnym odcinku opisane zostaną standardowe komunikaty FORTHA, słowo **ROLL**, a także kolejne błędy fig-FORTHA 1.1 i sposób ich usunięcia.

**MICHAŁ CHOROSZUCHA**

[1]) Standard FORTH79 przewiduje podawanie zamiast parametru LAST w słowie DUMP długości wyświetlanego obszaru w bajtach. Pozwala to na oglądanie słów FORTHA za pomocą następującej sekwencji:
' name length DUMP
dla obejrzenia pola parametrów, lub:
' name NFA length DUMP
dla całej definicji. Dla otrzymania takiej implementacji słowa DUMP należy zmienić trzecią linię kadru 241 (wydruk 2) na:
BASE @ >R HEX OVER + >R CR

**LITERATURA**

[1] fig-FORTH INSTALLATION MANUAL. GLOSSARY, MODEL, EDITOR. Release 1 with compiler security and variable length names — by William F. RAGSDALE November 1980, provided through the courtesy of the FORTH INTEREST GROUP Po Box 1105, San Carlos, CA 94070, USA

[2] FORTH, An Application-Oriented Language, REFERENCE MANUAL. Writer: Elizabeth D. Rather, Editors: Carolyn A. Rosenberg, Sue Linstrot. FORTH, Inc. 2309 Pacific Coast Highway, Hermosa Beach, CA 90254, USA, 1979—1980.

## *Samotesty*

# IV/B. Rzetelność danych

**Samotesty, opracowane i opublikowane przez Association for Computing Machinery *), nie stanowią standardu egzaminacyjnego i są przeznaczone wyłącznie do samodzielnego sprawdzenia własnej wiedzy.**

**PYTANIA**

9. Które z wymienionych niżej pojęć **nie ma** bezpośredniego związku z badaniem poprawności danych w zbiorze wejściowym?
   a) suma kontrolna (ang. check sum)
   b) liczba przerwań programu (ang. interruption count)
   c) suma umowna (ang. hash total)
   d) liczba rekordów (ang. record count)
10. Wymień kilka testów weryfikacyjnych stosowanych w badaniu rzetelności (ang. integrity) danych wejściowych.
11. Uzupełnij poniższe zdanie najwłaściwszym z wymienionych określeń: „Dla celów weryfikacji można wyliczyć ..... — z dołączonych przy nazwiskach pracowników numerów identyfikacyjnych — w fazie poprzedzającej właściwe sporządzanie listy płac".
    a) liczbę rekordów (ang. record count)
    b) sumę umowną (ang. hash total)
    c) cyfrę kontrolną (ang. check digit)
    d) saldo poziome (ang. crossfoot total)
12. Uzupełnij właściwą liczbą następujące zdanie: „Rozszerzając kod numeryczny o dodatkową cyfrę, wyrażającą sumę cyfr modulo 11, zapewniamy sobie wykrywanie ...% ogółu błędów powodowanych przestawieniem dwu cyfr".
    a) 0 b) 50 c) 90 d) 100
13. Do jakich z wymienionych niżej celów można w rachunkowości wykorzystywać tzw. pliki kontrolne (ang. audit files) albo dzienniki transakcyjne (ang. journals)?

*) Self-Assessment Procedure IV. Program Development Tools and Methods. Part B. Data Integrity. Communications of the Association for Computing Machinery, vol. 21, No. 2, February 1978, pp. 112—114.